UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA - PPGCAP

JACKSON MACHADO

SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS ATIVIDADES NA ÁREA DE MANUTENÇÃO DE SOFTWARE

JOINVILLE
2023

JACKSON MACHADO

SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS ATIVIDADES NA ÁREA DE MANUTENÇÃO DE SOFTWARE

Dissertação apresentada ao Programa de Pós–Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, como requisito para a obtenção do grau de Mestre em Computação Aplicada.

Orientadora: Avanilde Kemczinski
Coorientadora: Rebeca Schroeder Freitas

JOINVILLE
2023

Machado, Jackson
SMAE - Framework para diagnóstico das atividades na área de Manutenção de Software / Jackson Machado. - Joinville, 2023.
160 p. : il. ; 30 cm.

Orientadora: Avanilde Kemczinski.
Coorientadora: Rebeca Schroeder Freitas.
Dissertação (Mestrado) - Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas - CTT, Programa de Pós-Graduação em Computação Aplicada - PPGCAP, Joinville, 2023.

1. Engenharia de Software. 2. Qualidade de Software. 3. Manutenção de Software. 4. Métricas. 5. Framework. I. Kemczinski, Avanilde. II. Freitas, Rebeca Schroeder. III. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas - CTT, Programa de Pós-Graduação em Computação Aplicada - PPGCAP. IV. SMAE - Framework para diagnóstico das atividades na área de Manutenção de Software.

JACKSON MACHADO

SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS ATIVIDADES NA ÁREA DE MANUTENÇÃO DE SOFTWARE

Dissertação apresentada ao Programa de Pós–Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, como requisito para a obtenção do grau de Mestre em Computação Aplicada.

Orientadora: Avanilde Kemczinski
Coorientadora: Rebeca Schroeder Freitas

BANCA EXAMINADORA:

Avanilde Kemczinski
UDESC — Unversidade do Estado de Santa Catarina

Membros:

Roberto Pereira
UFPR — Unversidade Federal do Paraná

Isabela Gasparini
UDESC — Unversidade do Estado de Santa Catarina

Joinville, 14 de dezembro de 2023

Dedico este trabalho aos meus pais, a minha família, meus amigos e a todos os professores que me acompanharam e auxiliaram na realização desta importante etapa em meus estudos. Grato por tudo.

AGRADECIMENTOS

Agradeço à Deus por abençoar e guiar meu caminho, me dando forças durante essa longa jornada, permitindo que eu cresça e aprenda cada vez mais.

Agradeço as minhas orientadoras, Professora Doutora Avanilde Kemczinski e Professora Doutora Rebeca Schroeder Freitas, por toda atenção, carinho, direcionamento, condução, respeito, compreensão e amizade, por aceitarem esta tarefa de me guiar por esta nova etapa. Obrigado por acreditarem em mim, confiarem no meu trabalho e principalmente obrigado por me incentivarem a ir cada vez mais longe.

Agradeço ao Professor Doutor Roberto Pereira e à Professora Doutora Isabela Gasparini pela avaliação da dissertação de mestrado e pela colaboração fundamental na sua construção. Suas contribuições foram de inestimável importância para o desenvolvimento deste trabalho acadêmico.

Agradeço a todos os professores do Programa de Pós-Graduação em Computação Aplicada da Universidade do Estado de Santa Catarina - Udesc, pelo excelente trabalho que realizam e pela disponibilidade demonstrada. Agradeço por compartilharem generosamente seus conhecimentos e por toda a atenção que dispensaram ao longo deste processo.

Agradeço aos meus pais pelo apoio, carinho e atenção, estando disponíveis sempre que necessário e por prestarem todo auxílio possível. Obrigado por terem me ensinado a ser quem sou e por estarem sempre ao meu lado.

Aos meus dois filhos Bernardo e Aurora, por todo amor incondicional que tenho por vocês e que vocês demonstram por mim. Vocês são a minha base de vida e hoje sei que sou completo por ter vocês em minha vida.

Por fim, a todos os que, direta ou indiretamente, contribuíram para a realização deste trabalho, o meu sincero agradecimento.

“Não divido o mundo entre os fracos e os fortes,
ou entre sucessos e fracassos [...] divido o
mundo entre os que aprendem e os que não
aprendem.” (Benjamin Barber)

RESUMO

A área de manutenção de *software* é uma das mais antigas na engenharia de *software*, desempenhando um papel fundamental em todo o processo de desenvolvimento, tendo um papel crítico na garantia da qualidade do produto final e na estabilidade do mercado. No entanto, o controle das métricas de manutenção de *software* é uma tarefa desafiadora, uma vez que não existe um conjunto pré-definido de métricas de desempenho estabelecido para essa área. A decisão de quais métricas aplicar fica a critério das equipes de manutenção de *software*. Além disso, das perspectivas de manutenção de *software*, apenas a perspectiva de produto é diretamente utilizada pela indústria, com pouco proveito das perspectivas de processo e recursos. Esta pesquisa propôs a implementação de um *framework* para apoiar a tomada de decisão dos gestores na área de manutenção de *software* no gerenciamento das atividades, utilizando métricas de manutenção de *software* de maneira equilibrada entre as perspectivas de produto, processo e recurso. Além disto, considerou a dinâmica da área de manutenção de *software* e suas diversas atividades. Para atingir esse objetivo, utilizou-se a metodologia *Design Science Research Methodology* para criar um artefato computacional (um sistema web) que atenda o rigor científico. Realizou-se uma revisão sistemática da literatura para identificar os principais estudos sobre métricas de manutenção de *software* e uma pesquisa exploratória para entender como a indústria as aplica. Isso confirmou a lacuna na utilização de métricas de processo e recurso, bem como a complexidade e dinamismo das atividades, que tornam a gestão dessa área uma tarefa desafiadora. Após isso, o *framework* foi aplicado em um ambiente controlado de manutenção de *software* para avaliar sua utilização com gestores da área. Ademais, foi aplicado um questionário TAM para avaliação da aceitação do *framework*. O *framework* implementado demonstrou ser uma ferramenta aplicável para o gerenciamento da área de manutenção de *software*, auxiliando os gestores na alocação de recursos e na priorização das atividades conforme as necessidades da área, considerando seus produtos, processos e recursos.

Palavras-chave: manutenção de *software*, métricas, perspectivas, *framework*.

ABSTRACT

The software maintenance area is one of the oldest in software engineering, playing a fundamental role throughout the development process, with a critical role in ensuring the quality of the final product and market stability. However, controlling software maintenance metrics is a challenging task, as there is no predefined set of performance metrics established for this area. The decision of which metrics to apply is at the discretion of software maintenance teams. Furthermore, from the perspective of software maintenance, only the product perspective is directly used by the industry, with a few benefits from the process and resource perspectives. This research proposes implementing a framework to support decision-making by managers in the software maintenance area in managing activities using software maintenance metrics in a balanced manner among product, process, and resource perspectives. It also considered the dynamics of the software maintenance area and its various processes. To achieve this goal, the Design Science Research Methodology was used to create a computational artifact (a web system) that meets scientific rigor. A systematic literature review was conducted to identify the primary studies on software maintenance metrics, and an exploratory survey was carried out to understand how the industry applies them. This confirmed the gap in the use of process and resource metrics, as well as the complexity and dynamism of activities, making the management of this area a challenging task. Afterward, the framework was applied in a controlled software maintenance environment to assess its use with managers. Additionally, a TAM questionnaire was applied to evaluate the acceptance of the framework. The implemented framework proved to be an applicable tool for managing the software maintenance area, assisting managers in resource allocation and prioritizing activities according to the needs of the area, considering its products, processes, and resources.

Keywords: software maintenance, metrics, perspectives, framework.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACID	*Atomicity, Consistency, Isolation, Durability* (Atomicidade, Consistência, Isolamento, Durabilidade)
API	*Application Programming Interface* (Interface de Programação de Aplicativos)
ASP	*Adjusted Story Points* (Pontos de História Ajustados)
BDD	*Domain Complexity of the System* (Complexidade de Domínio do Sistema)
CBO	*Coupling Between Object* (Acoplamento entre Classes)
CC	*Class Complexity* (Complexidade da Classe/Tamanho)
CE	Critérios de Exclusão
CI	Critérios de Inclusão
CMMI	*Capability Maturity Model Integration* (Modelo de Capacidade e Maturidade Integrado)
CONF	Confiança na resolução de tarefas
D	Dificuldade de Desenvolvimento
DEVMSYS	Experiência de desenvolvimento de módulo
DEVSYS	Experiência total de desenvolvimento
DIT	*Depth of Inheritance Tree* (Profundidade na Árvore de Herança)
DOC	Atualização de Documentação
DSRM	*Design Science Research Methodology*
E	Esforço de Trabalho
ESS	Experiência com software
FPL	Familiaridade com a linguagem de desenvolvimento
GUT	Gravidade, Urgência e Tendência
IEC	*International Electrotechnical Commission*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ISO	*International Organization for Standardization*
LCOM	*Lack of Cohesion in Methods* (Falta de Coesão de Métodos)
LOC	*Lines of Code* (Linhas de Código)
MEXPAPP	Experiência de manutenção na aplicação
MEXPTOT	Experiência total de manutenção

MI	*Maintainability Index* (Índice de Manutenibilidade)
N	*Program Length* (Tamanho do Programa)
NDA	Número de dependências de precedência entre atividades
NDWP	Número de dependências entre produtos de trabalho e atividades
NDWPIn	Número de dependências de entrada dos produtos de trabalho com as atividades no processo
NDWPOut	Número de dependências de saída dos produtos de trabalho com as atividades no processo
NOA	*Number of Attributes* (Número de Atributos)
NOC	*Number of Children* (Número de Filhos na Árvore de Chamada)
NPR	Número de funções que participam do processo
PICO	*Population, Intervention, Comparison and Outcome* (População, Intervenção, Comparação e Resultados)
QP	Questão Principal
QS	Questão Secundária
RFC	*Response for a Class* (Respostas para a Classe)
RSL	Revisão Sistemática da Literatura
SES	Conformidade com os padrões de engenharia de software
SGBD	Sistema de gerenciamento de banco de dados
SMAE	*Software Maintenance Area Evaluation*
SPICE	*Software Process Improvement and Capability Determination*
SW-CMM	*Capability Maturity Model for Software* (Modelo de Maturidade de Capacidade para Software)
TAM	*Techology Acceptance Model* (Modelo de Aceitação de Tecnologia)
TCLE	Termo de Consentimento Livre e Esclarecido
TST	*Testability* (Testabilidade)
UDESC	Universidade do Estado de Santa Catarina
UFPR	Universidade Federal do Paraná
WMC	*Weighted Method per Class* (Método Ponderado por Classe)

SUMÁRIO

1 INTRODUÇÃO

A área de engenharia de *software* vem sofrendo grandes mudanças estruturais no que tange a gerência de *software*. Em especial, a área de manutenção de *software* vem se ajustando as novas realidades e a gestão acaba sofrendo um grande desgaste por não existirem indicadores definidos para identificar a real dinamicidade desta área (PRESSMAN, 2011).

Conforme aponta Sommerville (2011), a manutenção de *software* faz parte do ciclo de vida de desenvolvimento de *software* ou sistemas, estando presente após a liberação para uso. A área de manutenção é a responsável por realizar mudanças no *software* ou sistema que podem ser desde simples correções de erros de codificação, até mudanças mais complexas como alterações de requisitos e correções de erros de projeto.

Não distante disso, Pressman (2011) menciona que independentemente do domínio de aplicação, tamanho ou complexidade, o *software* continuará a evoluir com o tempo. Estas mudanças ocorrem quando são corrigidos erros, na necessidade de adaptação a um novo ambiente, por solicitação de novas características ou funções e quando a aplicação passa por um processo de reengenharia para proporcionar benefício em um novo contexto.

O termo "manutenção de *software*"vem sendo substituído, ou utilizado em conjunto com "evolução de *software*", como aponta Wazlawick (2013), pois, o termo evolução tende a ser mais adequado, haja visto que o intuito da área não é apenas manter o *software* como ele está, mas sim evoluí-lo para adaptar-se a novos requisitos. Não diferente disso, Jain, Tarwani e Chug (2016) destacam que durante toda a vida útil do *software*, ele é modificado continuamente por causa da evolução na tecnologia e crescente concorrência no setor.

Os autores Lehman e Belady (1985) formularam ao longo de décadas as leis que tratam sobre a evolução do *software*. Desta forma, eles mencionam que deve haver equilíbrio entre a necessidade de novas funcionalidades e a necessidade de desaceleração de mudança. Logo, a mudança contínua deve ser aplicada ou o *software* se torna progressivamente menos útil ao ambiente onde está implantado.

Outro ponto importante é que geralmente a implantação de funcionalidades após o lançamento do *software* se torna mais caro, conforme destaca Sommerville (2011), isso se dá devido à perda da estabilidade da equipe, seja pelas pessoas remanejadas a novos projetos, o corte de custos ou até mesmo aceleração de entregas. Logo, a má prática de desenvolvimento se torna evidente e a equipe de manutenção acaba necessitando de uma maior qualificação. O mesmo é destacado por Schnappinger et al. (2019), no qual mencionam que os custos de manutenção aumentam à medida que a base de código torna-se maior e os erros são propagados.

Por fim, Sommerville (2011) destaca também que, com o avançar da idade do programa, sua estrutura tende a se degradar, tornando sua compreensão mais difícil. Além disso, Hirama (2011) menciona que os custos de manutenção tende a crescer exponencialmente em relação ao custo de desenvolvimento, caso não sejam aplicados os conceitos e técnicas adequadas no desenvolvimento de *software*.

Desse modo, realizar medições que avaliem o desempenho da área de manutenção é parte fundamental do processo, para garantir, com isso, o pleno funcionamento da área, bem como, segundo Ostberg e Wagner (2014), manter o *software* sustentável economiza tempo e dinheiro. Os autores Fenton e Pfleeger (2004) determinam que medições são processos que atribuem valor a certas propriedades, de maneira a poder descrevê-las. Com isso, pode-se definir que na engenharia de *software*, essas propriedades são objetos ou eventos, que podem ser classificados em produtos, processos e recursos.

Os autores Meidan et al. (2018) apontam, que as constantes evoluções de processo trazem uma grande dificuldade para as medições, pois não há um conjunto pré-definido de métricas de desempenho para a área de manutenção de *software*, o que tornam as pesquisas nessa área cansativas e complexas, demonstrando a importância das definições de indicadores que avaliem o desempenho da equipe de manutenção.

Entretanto, devido à inúmera quantidade de artigos de medição de *software* disponíveis, trazendo consigo uma grande quantidade de métricas e variáveis, a tarefa de sua sumarização e estruturação é um grande desafio. Tratando especificamente de manutenção de *software*, por não haver estudos direcionados especificamente para essa área, tem-se a carência de pesquisas mais aprofundadas sobre este assunto (MEIDAN et al., 2018).

O presente trabalho apresenta uma revisão sistemática da literatura cuja finalidade é compreender o estado da arte sobre métricas de manutenção de *software*. Este mapeamento auxiliou na identificação das principais métricas levantadas pela literatura, o que permitiu compreender como a literatura explicita como as métricas na área de manutenção de *software* devem ser aplicadas. Para corroborar com a revisão sistemática da literatura desenvolvida, também foi realizada uma pesquisa exploratória. O intuito desta pesquisa foi compreender como a indústria utiliza (e se utiliza) as métricas identificadas na revisão.

De maneira a estruturar a pesquisa científica deste trabalho, foi aplicado o *Design Science Research Methodology* que direciona a produção de um artefato computacional, sendo que sua aplicação será melhor detalhada durante a Seção 2.5.

Logo, a partir desta investigação, desenvolveu-se um artefato computacional com o propósito de auxiliar os gestores da área de manutenção de *software* na alocação de seus recursos. Esse artefato considerou as três perspectivas (produto, processo e recurso) e as diversas atividades realizadas pela área de manutenção de *software*, sendo viabilizado por meio de um *framework* que disponibilizou as métricas de manutenção de *software* respaldadas pela literatura e pela indústria, conforme identificado durante o Capítulo 3, com o objetivo recomendar a alocação de recursos.

1.1 PROBLEMA

A área de manutenção de *software* caracteriza-se por um grande dinamismo, uma vez que a entrada constante de tarefas decorrente dos produtos de mercado gera um impacto significativo

no planejamento de entregas. Nesse contexto, não se trata de um projeto fechado de desenvolvimento, mas sim de uma entrada contínua de atividades, resultando em imprevisibilidade no atendimento.

Este dinamismo se dá muito porque que as informações não são estáveis, conforme apontam Hunt e Thomas (2010). A compreensão de um requisito muda conforme a quantidade de informações que se tem sobre ele. Dessa forma, é inegável que, conforme as informações surgem, a compreensão muda. Não somente, Hunt e Thomas (2010) afirmam que a manutenção não pode ser uma atividade isolada, mas sim deve fazer parte rotineira do processo de desenvolvimento, isso se dá justamente por esta dinâmica informacional.

Não longe disto, a falta de compreensão por parte do usuário referente a sua real necessidade, ligada com as mudanças frequentes de prioridades e requisitos estão diretamente ligadas a dinâmica informacional, para isso Jain, Tarwani e Chug (2016) afirmam que 65% do esforço gasto em manutenção é usado para implementar as alterações feitas no requisito. Paduelli (2007) afirma que este é um dos principais problemas da área de manutenção de *software* e, além disso, a falta de metodologias e procedimentos de manutenção e a dificuldade na medição do desempenho da equipe também são problemas latentes da área. Estes problemas estão ligados ao fato de que a manutenção não é tratada como parte rotineira do processo de desenvolvimento.

Logo, a identificação de uma normalização dos indicadores ligados especificamente a manutenção de *software*, permitiria identificar quais são as variáveis que constituem esses indicadores e, com isso, definir quais os pontos na área de manutenção de *software* que mais influenciam no desempenho da equipe de manutenção. Esta é a base de muitas iniciativas como SW-CMM (*Capability Maturity Model for Software*), ISO/IEC 15504 (*SPICE, Software Process Improvement and Capability Determination*) e CMMI (*Capability Maturity Model Integration*), que tentam apontar indicadores normalizados para a área geral de engenharia de *software*.

Com base nisto, é possível destacar que a área de manutenção de *software* apresenta uma carência de padronização das métricas aplicadas, pois atualmente não foram identificados padrões de métricas aplicadas a essa área e devido a isto, não há como comprovar seu real gasto, bem como não é possível identificar os reais impactos de um "replanejamento" de tarefas (MOLNAR; NEAMTU; MOTOGNA, 2020). Neste mesmo ponto, Calero, Bertoa e Moraga (2013) destacam que embora a manutenção seja uma prática padronizada em várias das disciplinas de engenharia, não existe atualmente tal consciência na comunidade de engenharia de *software*.

Para mitigar estas problemáticas, cabe uma melhoria nas ferramentas de avaliação da área de manutenção de *software*, pois as atuais ferramentas não suprem todas as atividades que essa área necessita. Isto se dá, devido a esta falta de padronização das métricas e até mesmo falta de ferramentas que auxiliem a efetiva avaliação da área de manutenção. A problemática desta investigação de mestrado visou discutir a importância de realizar as medições adequadas na área de manutenção de *software*, o que permite a compreensão de suas necessidades e o reflexo no desempenho da equipe, buscando responder à questão: "Como apoiar os gestores na tomada de decisão da priorização e alocação das atividades de manutenção de *software*, de forma

automatizada?”.

1.2 JUSTIFICATIVA

A padronização das métricas de manutenção de *software* visa demonstrar qual o melhor caminho para avaliação da área de manutenção de *software*. Ao se compreender quais as principais métricas trazidas pela literatura e quais as principais métricas utilizadas pela indústria, é possível determinar quais métricas são aplicadas nas atividades da manutenção de *software*, permitindo, com isso, determinar as alocações de recursos para uma determinada atividade, mitigando os problemas com a dinâmica da área e acelerar o desenvolvimento de *software*.

Para esta questão, como abordado anteriormente, esta investigação apresenta duas pesquisas, sendo uma revisão sistemática da literatura, cujo objetivo é demonstrar o estado da arte em relação a métricas de manutenção de *software*, e uma pesquisa exploratória, cujo objetivo é correlacionar o que foi identificado na revisão sistemática da literatura e o aplicado na indústria.

Com isso, a contribuição científica que se buscou com esta pesquisa está na compreensão atual da área de manutenção de *software*, que visa determinar as principais métricas de manutenção de *software*, criando com isso um *framework* que apoie os gestores de manutenção de *software* na decisão de alocação de seus recursos nas diversas atividades da área de manutenção de *software*.

1.3 OBJETIVOS

A otimização das entregas na área de manutenção de *software*, visa permitir que uma empresa possa reduzir seu tempo e custo, bem como fazer um melhor uso de seus recursos durante o processo de manutenção de *software*, sendo estes os motivadores que geraram a pergunta desta pesquisa. Para responder esta pergunta, foram definidos objetivos para orientar o processo de pesquisa. O objetivo geral é apresentado na Seção 1.3.1 e os objetivos específicos são apresentados na Seção 1.3.2.

1.3.1 Objetivo Geral

O objetivo principal desta dissertação de mestrado consiste na implementação de um *framework* que tem por finalidade oferecer suporte aos gestores da área de manutenção de software, auxiliando-os na tomada de decisões referentes à priorização e alocação de atividades de seus colaboradores (ou equipe).

1.3.2 Objetivos Específicos

Visando apoiar o alcance do objetivo geral, foram definidos objetivos específicos para as etapas deste trabalho, sendo estes:

- Investigar a teoria e estado da arte sobre as métricas de manutenção de *software*, buscando um padrão de métricas aplicável;
- Investigar as métricas aplicáveis atualmente na indústria de *software*, em especial na área de manutenção de *software*, realizando uma correlação teórica-prática;
- Desenvolver uma ferramenta para aplicação do *framework*;
- Aplicar e avaliar a ferramenta no contexto empresarial.

1.4 ESCOPO

A manutenção de *software* é uma das principais áreas do desenvolvimento de *software*, onde ocorre o processo de adaptação e otimização de um *software* já desenvolvido, bem como a correção de defeitos que ele possa ter (WAZLAWICK, 2013). Sua aplicação se faz necessária para o produto poder preservar sua qualidade ao longo do tempo, pois, caso não seja realizada, haverá uma deterioração do valor percebido desse *software*, seja por conta de uma falta de evolução tecnológica ou de evolução conceitual, por conta disso, Schnappinger et al. (2019) afirma que existe uma relação direta entre a manutenibilidade de um sistema e sua lucratividade.

Para ocorrer uma evolução de uma determinada área, é natural haver a necessidade de avaliações e medições que permitam identificar onde estão os pontos de melhoria. Não diferente disso, a área de manutenção de *software* também precisa ser aprimorada.

Para alcançar o objetivo desta pesquisa, foi implementado um *framework* denominado como *Software Maintenance Activity Evaluation* (SMAE), sendo um *framework* de apoio a decisão que permite aos gestores da área de manutenção de *software* tomarem decisões assertivas em relação ao processo de manutenção de *software*, aos recursos alocados e os produtos oferecidos. Desse modo, o *framework* fornece suporte a decisão dos gestores na priorização e alocação de recurso das atividades da área de manutenção de *software*, e permite definir a complexidade de manutenção de uma determinada atividade e quais os principais recursos a serem alocados.

1.5 METODOLOGIA

A metodologia utilizada pode ser classificada como uma pesquisa de natureza aplicada, haja visto que planeja gerar conhecimento por meio da aplicação prática da solução em um determinado problema, se utilizando de conhecimentos prévios e fundamentação teórica baseada na bibliografia levantada (GIL, 2017).

Sobre os objetivos, a pesquisa pode ser classificada como exploratória, conforme Wazlawick (2010), por buscar um conjunto de métricas que melhor determinem a dinâmica da área de manutenção de *software*, seu objetivo foi desenvolver, esclarecer e modificar conceitos e ideias ainda em fase inicial, realizar a formulação de problemas mais direcionados para estudos posteriores (GIL, 2008).

Para guiar o desenvolvimento do *framework* proposto, foi utilizada a abordagem de *Design Science Research Methodology*, sendo uma metodologia cujo objetivo é legitimar o desenvolvimento de artefatos como um meio para a produção de conhecimentos científicos do ponto de vista epistemológico (PIMENTEL; FILIPPO; SANTORO, 2020).

Há seis etapas específicas para a aplicação do *Design Science Research Methodology*, sendo estas a Identificação do Problema e Motivação, Definição dos Objetivos da Solução, Projeto e Desenvolvimento, Demonstração, Avaliação e Comunicação, estas etapas são melhor abordadas na Seção 2.5 do Capítulo 2 e no Capítulo 4.

Ainda, para a etapa de Identificação dos Problema e Motivação, foi utilizado o procedimento metodológico de pesquisa bibliográfica para estruturação dos conceitos de processo geral de *software*, manutenção de *software*, qualidade de *software* e medições utilizadas na área de manutenção de *software*. Os trabalhos relacionados foram pesquisados por meio de uma revisão sistemática da literatura (RSL) realizada conforme o protocolo definido por Petersen et al. (2008).

Ademais, com base nos resultados dos trabalhos relacionados, foi realizado um segundo procedimento metodológico mediante uma pesquisa exploratória, baseada no protocolo definido por Ciolkowski et al. (2003), com os gestores da área de manutenção de *software*, cujo objetivo foi investigar a correlação das métricas trazidas pela literatura e as métricas atualmente aplicadas na indústria.

Com os resultados da pesquisa bibliográfica e da pesquisa exploratória, foi desenvolvido um *framework* para fornecer suporte aos gestores na priorização e alocação de recursos nas atividades da área de manutenção de *software*, sugerindo alocações de recurso com base nestas atividades.

A partir do *framework* SMAE desenvolvido, foram realizadas avaliações para verificar se o mesmo está aderente as práticas de medições da área de manutenção de *software*, com o objetivo de auxiliar na alocação ou realocação de recursos durante o processo de manutenção de *software*, auxiliando na tomada de decisões durante o ciclo de desenvolvimento de *software*. A avaliação do *framework* foi realizada em ambiente real de desenvolvimento de *software*, onde a percepção de utilidade e aceitação foi coletada através da aplicação de um questionário que seguiu o modelo TAM, que é abordado no Capítulo 2 e Capítulo 6.

1.6 ESTRUTURAÇÃO

Esta pesquisa inicia com a fundamentação teórica, que visa descrever os principais tópicos no capítulo 2: processo geral de *software*, manutenção de *software*, qualidade de *software* e medições da área de *software*.

No capítulo 3 são apresentados os trabalhos relacionados às medições de *software* identificadas por meio da revisão sistemática da literatura e da pesquisa exploratória realizada para este trabalho.

No capítulo 4 é apresentado o *framework* desenvolvido neste trabalho de pesquisa. No capítulo 5 é apresentado o experimento e a avaliação da ferramenta. Por fim, no capítulo 6 é apresentada as conclusões desta investigação e trabalhos futuros, seguida das referências bibliográficas, apêndices e anexos.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica visa abordar os principais assuntos relacionados a proposta de pesquisa. Inicialmente é introduzido o conceito de processo geral de *software* na seção 2.1 que determina onde a manutenção de *software* se encontra no ciclo de vida de desenvolvimento de *software*, sendo que esta é detalhada na seção 2.2. A seção 2.3 busca discutir o motivo no qual a manutenção entra como um critério de qualidade de *software*, compreendendo então como as métricas são aplicadas e detalhadas na seção 2.4. A seção 2.5 detalha o processo metodológico utilizado para legitimar a construção do artefato como meio de produção do conhecimento científico. Por fim, o capítulo é encerrado com uma reflexão na seção 2.6 sobre como as métricas são necessárias para a área de manutenção de *software*.

2.1 PROCESSO GERAL DE *SOFTWARE*

Pode-se citar que o processo de *software* trata-se de um conjunto de atividades relacionadas à produção de um *software*, conforme Sommerville (2011), levando ao desenvolvimento de um produto a partir do zero em uma linguagem padrão de desenvolvimento. Nesta mesma linha, Pressman (2011) destaca que o processo de *software* é definido como uma metodologia para a execução das atividades, etapas e tarefas necessárias para desenvolver um *software* de qualidade.

Ainda, Wazlawick (2013) caracteriza um processo de *software* como um conjunto de atividades que são interdependentes, que possuem responsáveis e que tenham entradas e saídas definidas. Contudo, Meidan et al. (2018) vai além e destaca que o desenvolvimento de *software* é um processo longo, caro e complexo, sendo que o resultado deste processo não é apenas o produto final, mas também a produção de artefatos intermediários e suplementares durante o esforço de desenvolvimento.

Desta forma, se afirma que um processo de *software* deve definir como será a abordagem para a elaboração de um *software*, de maneira a executar atividades encadeadas que permitam a entrada e saída de artefatos.

Conforme destacam García-Borgoñón et al. (2014) e Sommerville (2011) os processos de *software* são mais complexos e imprevisíveis do que os processos de produção típicos, contudo, todos devem incluir quatro atividades fundamentais para a engenharia de *software*:

- **Especificação de *software*:** Definição das funcionalidade e restrições do *software*;
- **Projeto e implementação de *software*:** Produção do *software* atendendo as especificações;
- **Validação de *software*:** Validação para garantir o atendimento às especificações;
- **Evolução de *software*:** Evolução para atendimento às novas necessidades e mudanças do ambiente.

Com base nisso, Pressman (2011) também aborda que um processo não se trata de uma prescrição rígida, muito pelo contrário, trata-se de uma abordagem adaptável para se poder realizar o trabalho da melhor forma quanto possível, sempre entregando o *software* no prazo e com qualidade suficiente para satisfazer àqueles que patrocinaram a sua criação. Contudo, metodologias são essenciais para estabelecer o alicerce para um processo de engenharia de *software* completo, no qual permite a identificação de atividades estruturais para os processos.

Kuhrmann et al. (2016) aborda que o desenvolvimento de *software* é caracterizado por sua diversidade, logo definir a abordagem ideal para desenvolver *software* já é debatido a algum tempo. Como não existe "bala de prata", bem como os processos de *software* devem refletir as necessidades de projetos de *software* específicos, os processos de *software* precisam ser flexíveis e adaptáveis.

Outra característica importante é que, conforme afirma Sommerville (2011), os processos de *software* são complexos e dependem de pessoas para tomar decisões e fazer julgamentos, sendo que não existe um processo ideal para todas as organizações, cabendo a cada uma deles desenvolver e aplicar o que lhe for mais aderente. Entretanto, deve haver sempre espaço para melhoria, avaliação e revalidação destes processos, pois sempre é necessário aplicar as melhores práticas de engenharia de *software* disponíveis.

Conforme destaca Valente (2020), os sistemas modernos são desenvolvidos em equipes e, para que consigam produzir *software* com qualidade e produtividade, precisam de um mínimo ordenamento, desta forma, os processos são instrumentos que as empresas dispõem para coordenar, motivar, organizar e avaliar o trabalho de seus desenvolvedores.

A execução de um processo é chamada de projeto, e um processo é uma receita que é seguida durante a realização de um projeto, conforme cita Filho (2001), podendo dizer que o projeto é o empreendimento que concretiza uma abstração, que é o processo. Com isso, processos, pessoas e tecnologias são o que constituem os fatores de produção de um *software*, de maneira a entregar um produto de qualidade suficiente, em um prazo aceitável e com custos viáveis.

Por fim, não é possível obter melhoria sustentada até que o processo esteja sob controle estatístico, medir o processo de *software* é importante para monitorar, controlar, avaliar, gerenciar e propor melhorias (MEIDAN et al., 2018). Diante disso, Sommerville (2011) e Perkusich et al. (2015) apontam que os processos de *software* podem ser melhorados pela padronização, possibilitando uma melhora na comunicação, uma redução no tempo de treinamento e também torna o suporte automatizado e o processo mais econômico.

Logo, se conclui que a base de todo processo de *software* está na execução de atividades por recursos (pessoas) e que permitam a entrada e saída de artefatos, sendo que sua execução de forma lógica permite a construção de um produto final, a ser entregue para os usuários finais conforme as especificações levantadas. Por fim, todo e qualquer processo pode ser melhorado e avaliado para trazer um melhor desempenho em sua comunicação e agilidade das atividades.

2.1.1 Ciclo de Vida de Desenvolvimento de *software*

A construção de um *software*, conforme afirma Pressman (2011), trata-se de um processo de aprendizado social e iterativo, ou seja, é a incorporação do conhecimento coletado, filtrado e organizado conforme se desenvolve o processo. Além disso, Masso et al. (2020) afirmam que o *software* é o resultado de um processo que depende de uma boa gestão em cada uma de suas atividades.

Entretanto, há diferentes tipos de processos de *software*, sendo que cada um deles possui sua própria organização de atividades metodológicas, bem como as etapas e tarefas que ocorrem dentro de cada atividade em relação à sequência e ao tempo, esse aspecto é denominado fluxo de processo (PRESSMAN, 2011).

É comumente aceito que qualquer processo de *software* precisa ser adaptado aos requisitos do projeto em particular, de acordo com Pazin, Allian e Jr. (2018), é necessário que os engenheiros de *software* durante o gerenciamento, se utilizem de técnicas e ferramentas de apoio, de maneira a controlar a variabilidade desses processos.

Por essência, o processo inicial de adoção do desenvolvimento é o Codificar e Consertar, não se trata efetivamente de um modelo de fluxo de processo, mas sim de um "antimodelo"por excelência, pois é utilizado quando não se utiliza conscientemente nenhum modelo (BOEHM, 2006);(WAZLAWICK, 2013).

Com isso, os demais fluxos de processo podem ser determinados através de modelos genéricos, que não são descrições definitivas dos processos, conforme destaca Sommerville (2011), sendo abstrações que podem ser usadas para explicar as diferentes abordagens de desenvolvimento, denominado de *framework* do processo. Um modelo de processo de *software* é uma representação abstrata de um processo (BASEER, 2015).

Estes fluxos nada mais são do que a mudança de uma produção artesanal para a elaboração e desenvolvimento de uma forma com maior previsibilidade e qualidade, conforme destaca Wazlawick (2013). Estes modelos de processos por vezes são conhecidos como ciclos de vida sendo que existem duas grandes famílias de modelos, os prescritivos e os ágeis.

Os primeiros modelos foram parte fundamental da engenharia de *software*, conforme destacam Alam, Sarwar e Noreen (2022), pois forneceram uma estrutura para diferentes atividades de desenvolvimento de *software*, haja visto que, com o passar do tempo a programação tornou-se complicada, consequentemente surgindo a necessidade de atualizar a estrutura de desenvolvimento de *software*.

Os processos prescritivos, que também são conhecidos como processos dirigidos a planos, possuem suas atividades já planejadas com antecedência e o progresso é avaliado por comparação com o planejamento inicial (SOMMERVILLE, 2011).

Um dos principais modelos prescritivos é o *Waterfall* ou Cascata, sua principal características é a existência de fases bem definidas e sequências, conforme destaca Wazlawick (2013). Desta forma, todo seu planejamento é feito antes de sua efetiva execução, logo conforme suas

etapas ocorrem, as mesmas são avaliadas para que se tenha a verificação da efetividade da entrega e a previsão de seu encerramento.

Os autores Dima e Maassen (2018) determinam que o modelo em cascata assume um seguimento de sequências nas fases de desenvolvimento do *software*, começando com as instruções sobre os requisitos do cliente e posteriormente seguidas de sua implementação prática com a construção do produto.

Ainda, os modelos prescritivos não são mutuamente exclusivos, ou seja, podem ser aplicados em conjunto conforme as equipes, produtos e processos que serão utilizados, de acordo com Sommerville (2011) e Baseer (2015), isso ocorre principalmente em sistemas de grande porte que, por muitas vezes, fazem sentido combinar algumas das melhores características de cada modelo.

O autor Pressman (2011), afirma ser possível descrever uma metodologia de processo genérica para a engenharia de *software*, em que estabelece cinco atividades metodológicas: comunicação, planejamento, modelagem, construção e entrega. Com base nestas atividades, é possível determinar e exemplificar 4 (quatro) principais fluxos de processo, conforme exibido na Figura 1. Estes fluxos são:

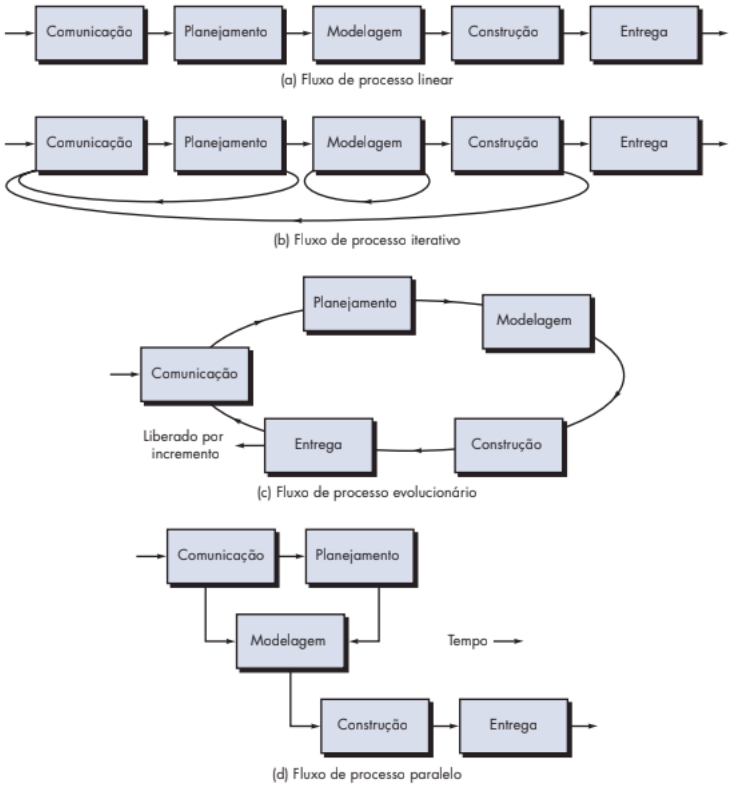
- **Linear:** Executa cada uma das atividades em sequência, começando com a comunicação e encerrando com a entrega;
- **Iterativo:** Repete uma ou mais das atividades antes de continuar para a próxima;
- **Evolucionário:** Executa as atividades de uma forma cíclica, sendo que a cada volta conduz a uma versão mais completa do *software*;
- **Paralelo:** Executa uma ou mais atividades em paralelo.

Ainda, Pressman (2011) destaca que os modelos de processos prescritivos foram propostos originalmente para trazer ordem ao caos no desenvolvimento de *software* e que estes proporcionaram uma considerável contribuição quanto à estrutura utilizável, entretanto, Pressman (2011) complementa que os trabalhos de engenharia de *software* e o seu produto permanecem à beira do caos, mesmo com a aplicação destes modelos.

Com isso, originaram-se os modelos ágeis, que trazem uma nova abordagem para a área de desenvolvimento de *software*, através de um planejamento gradativo e com uma maior facilidade na alteração dos processos, de maneira a refletir as necessidades de mudança dos clientes (SOMMERVILLE, 2011).

No caso de projetos onde os requisitos mudam rapidamente ou onde a concorrência no campo do produto é aumentada e novas funcionalidades precisam ser implementadas rapidamente, os modelos ágeis são considerados mais adequados devido à sua maior flexibilidade na implementação dos requisitos do cliente e lançamentos frequentes de produtos (DIMA; MAASSEN, 2018).

Figura 1 – Fluxo de Processo



Fonte: Pressman (2011)

Os métodos ágeis seguem uma filosofia diferente da filosofia dos modelos prescritivos, pois, ao invés de apresentarem fases bem definidas e estruturadas, eles se focam em valores humanos e sociais (BOEHM, 2006; WAZLAWICK, 2013).

Sua origem se dá em 2001, quando 17 (dezessete) renomados desenvolvedores e pesquisadores, entre eles Kent Beck, Martin Fowler, Alistair Cockburn e Robert Martin, assinaram o "Manifestado para o Desenvolvimento Ágil de *software*".

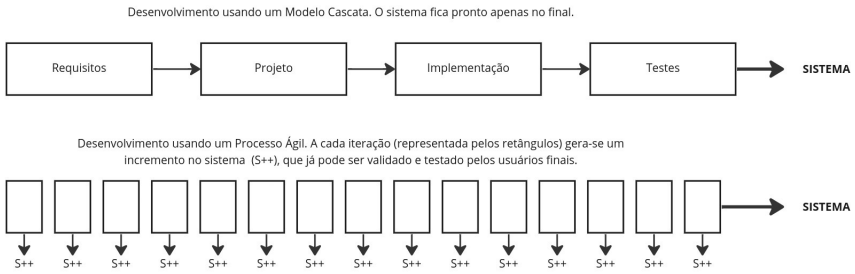
O manifesto ágil se baseia em princípios fundamentais, como o valor atribuído a indivíduos e interações, a entrega de *software* funcional, a colaboração contínua com o cliente e a capacidade de resposta à mudança. Além disso, o manifesto aborda questões relacionadas ao gerenciamento de conhecimento e experimentação. Ele tem aplicações amplas na educação, no ensino, no ambiente acadêmico e fornece motivação para equipes ágeis, facilitando a transição para práticas ágeis e promovendo o desenvolvimento de comunidades de prática no campo do desenvolvimento de *software* (BASEER, 2015).

O autor Pressman (2011) aponta que os métodos ágeis tentam sanar fraquezas reais e perceptíveis da engenharia de *software* convencional, oferecendo benefícios importantes. Em linhas gerais os métodos ágeis trouxeram para os processos de *software* a adaptabilidade, essencial para a resposta rápida as alterações do projeto. Esta metodologia promove a distribuição de funções e responsabilidades na equipe, incentiva a comunicação frequente entre clientes e

membros da equipe, e divide o trabalho em tarefas e entregas regulares (FLORIANI; STEIL, 2021).

Uma das principais características do processo ágil, conforme aponta Valente (2020), é a adoção de ciclos curtos e iterativos de desenvolvimento, por meio dos quais um sistema é implementado de forma gradativa, se iniciando com o que é mais urgente para o cliente. Uma representação dessa sistemática é apresentada na Figura 2.

Figura 2 – Comparação de Modelo Cascata com Modelo Ágil



Fonte: Adaptado de Valente (2020)

Além disso, Malhotra e Chug (2016) também trazem que as metodologias ágeis dão ênfase à comunicação face a face, bem como não envolve um planejamento de longo prazo, mas sim a criação de um modelo de trabalho que se adapte a alteração de requisitos rapidamente.

Entretanto, não há o que apontar como qual o melhor, ou ainda o único processo de desenvolvimento a ser seguido. Como já pontuado anteriormente, cabe a organização optar pela melhor abordagem para o seu desenvolvimento, conforme destaca Valente (2020). Geralmente, é necessário encontrar um equilíbrio entre os processos dirigidos a planos e os processos ágeis (WAZLAWICK, 2013).

Portanto, compreende-se que é responsabilidade da engenharia de *software* estabelecer fluxos de processos que possam servir como diretrizes para os procedimentos de *software* a serem implementados na organização. É incumbência da engenharia de *software* determinar se o processo principal será único ou se haverá a opção por um ou mais processos, de maneira a extrair o máximo potencial de sua equipe de desenvolvimento.

2.2 MANUTENÇÃO DE *SOFTWARE*

O ciclo de vida de *software* não encerra com a sua entrega, pois, conforme destaca Sommerville (2011), os mesmos perduram durante toda a vida útil do sistema, já que para o mesmo se manter útil, é inevitável que ocorram mudanças, seja na área de negócio ou na correção de algum defeito identificado.

Com isso, a manutenção de *software* nada mais é do que o processo de adaptação e otimização de um *software* já desenvolvido, bem como a correção de defeitos que ele possa ter (WAZLAWICK, 2013).

A manutenção de *software* é um grande desafio, pois deve-se enfrentar uma fila crescente de correções, solicitações de adaptação e novas funcionalidades no sistema, desta forma essa crescente quantidade de tarefas ameaça utilizar todos os recursos disponíveis, fazendo com que a organização gaste mais tempo e dinheiro com a manutenção do que com eventuais novos programas, conforme destaca Pressman (2011). Não somente isso, ele continua apontando que não é rara uma organização que despende de 60% a 70% de seus recursos com manutenção de *software*.

Na década de 1970, os custos foram em torno de 60%, enquanto nas décadas de 1990 e 2000, os custos relatados aumentaram para cerca de 90%. Assim, as organizações de desenvolvimento de software percebem que a transformação do trabalho de manutenção representa uma oportunidade para simplificar a gestão de muitas das tarefas diárias relacionadas à manutenção e, ao mesmo tempo, reduzir custos (ULZIIT et al., 2015).

Também os autores Polo et al. (1999) e Sun et al. (2015) apontam que a fase de manutenção é conhecida como a mais cara do ciclo de vida, sendo que a razão desse elevado custo está na natureza imprevisível desta atividade. Não somente isso, um dos principais desafios da área de manutenção se trata do curto tempo em que a equipe de manutenção é exigida para realização das modificações, atrelado a compreensão limitada que a mesma pode possuir por não fazer parte integrante da equipe original de desenvolvimento (BOURQUE; FAIRLEY, 2014).

O objetivo da manutenção de *software* não é apenas garantir o bom desempenho e operação desses sistemas, sustentando o produto de *software* ao longo de seu ciclo de vida, mas também agregar e adaptar novas tecnologias aos ambientes existentes (KHEZAMI; KESSENTINI; FERREIRA, 2021).

Corroborando com este aspecto, Pressman (2011) traz que muitas vezes os clientes também não sabem o que efetivamente desejam, e, mesmo que saibam, essas necessidades se modificam no decorrer do projeto, o que aumenta a imprevisibilidade.

Diante deste cenário de imprevisibilidade, a manutenção de *software* é diferente de acordo com o que o sistema pretende resolver, conforme aponta Pfleeger (2004), onde alguns sistemas têm maior tendência a sofrer mudanças do que outros, e isso conforme o nível de dependência que os requisitos do sistema tem do mundo real, sendo indicado pelo autor a existência de três tipos de sistemas:

- **Sistema S:** São sistemas fortemente definidos e derivados de uma especificação fechada. Normalmente são sistemas estáticos e que não se adaptam facilmente a mudanças, pois estão destinados à resolução de um problema específico. Um exemplos de sistema S seria um sistema que propõem soluções matemáticas, pois se tratam de especificações fechadas e já definidas, sendo necessária apenas a sua implementação;

- **Sistema P:** São sistemas desenvolvidos com base em especificações de requisitos abstratos, derivados de uma abstração prática de um problema a ser resolvido. Os requisitos desses sistemas são formulados com base em aproximações, e as soluções produzidas

são influenciadas pelo ambiente no qual serão implementadas. Um exemplo de sistema P é um jogo de xadrez, onde todas as regras já são especificadas e definidas, contudo a cada nova jogada há inúmeras possibilidade e cálculos a serem considerados, desta forma desenvolve-se uma solução aproximada para que seja mais prática a construção e utilização;

- **Sistema E:** São sistemas embutidos no mundo real e que se modificam conforme o mundo muda, tendo como solução um modelo de processo abstrato que é parte integrante do mundo em que está envolvido. Um exemplo de sistema E seria um previsor de estabilidade econômica em um país, como há mudanças no mundo e a economia se movimenta conforme essas mudanças, o sistema deve se adaptar a essas alterações, sendo necessária sua modificação.

Considerando estes três tipos de sistema, a manutenção e evolução de *software* sempre se fará mais presente naquele que possuir maior proximidade do mundo real, pois irá se modificar conforme o mundo real se modifica. Por exemplo, um Sistema S tende a ser mais estável, pois resolve um problema de uma forma rígida, já um Sistema P tente a sofrer modificações, pois a abstração aplicada pode sofrer uma nova interpretação ou ainda o ambiente aplicado pode se modificar, por fim, o Sistema E possivelmente estará em constante evolução, pois é construído para ser maleável.

Outro ponto importante da manutenção de *software*, como já mencionado anteriormente, está no custo que a mesma terá. Pode-se afirmar que geralmente a implantação de funcionalidades após o lançamento do *software* se torna mais caro (SOMMERVILLE, 2011). Varga (2018) menciona que os motivos que fazem este custo ser mais caro são a dificuldade em adaptar o *software* a uma nova funcionalidade, no qual não estava originalmente prevista, bem como a rotatividade de pessoas, que podem ser remanejadas de projeto ou saírem da organização, fazendo com que o projeto esteja suscetível a má prática de desenvolvimento, bem como a equipe de desenvolvimento irá carecer de qualificação. Por fim, (VARGA, 2018) complementa que com o avançar da idade do programa, sua estrutura tende a se degradar, tornando sua compreensão mais difícil.

Pressman (2011) afirma que uma das principais razões dos problemas na manutenção de *software* está na mobilidade dos profissionais, pois, é bem provável que a equipe responsável pelo trabalho original já não esteja mais alocado a este projeto, cabendo a outra geração de profissionais conduzi-lo. Sommerville (2011) indica que para a solução de problemas como estabilidade da equipe, a má prática de desenvolvimento e qualificação pessoal pode ser resolvida pelas organizações quando aceitarem que a manutenção faz parte do ciclo de vida e não é apenas uma atividade separada e de segunda classe. Já para a idade avançada de um programa, o mesmo destaca que há processos de reengenharia e *refactoring* que podem ser aplicados para a melhoria no decorrer do tempo de vida do *software*.

Para tentar sanar esta imprevisibilidade da manutenção de *software*, os autores Lehman e Belady (1985), por meio dos estudos da dinâmica da evolução de programas, propuseram as 'Leis de Lehman' que tratam sobre a evolução do *software*, mencionando que deve haver equilíbrio entre a necessidade de novas funcionalidades e a necessidade de desaceleração de mudança. Desta forma, indicam que a mudança contínua deve ser aplicada ou o *software* se torna progressivamente menos útil ao ambiente onde está sendo utilizado, devendo haver parcimônia entre novas funcionalidades e a manutenção das atuais.

Nabilah e Sunindyo (2019) determinam que a evolução do *software* refere-se a mudanças dinâmicas nas características e comportamentos do *software* trazidas através da atividade de manutenção de *software*, sendo que as "Leis de Lehman" são ideias de mudanças sustentáveis, que descrevem uma série de *insights* derivados de estudos de longo prazo de evolução do sistema.

As "Leis de Lehman" visam estabelecer que a evolução de *software* é inevitável, contudo, mesmo para esta evolução, são impostos limites em relação ao que as equipes de manutenção podem fazer. Um exemplo disso está na lei que determina a autorregulação, que impede que o trabalho demasiado ou insuficiente seja executado, pois pode acarretar descontinuação do sistema (WAZLAWICK, 2013). Desta forma, Lehman e Belady (1985) trouxeram as leis destacadas no Quadro 1.

Quadro 1 – Leis de Lehman

(continua)

Mudança contínua	Deve haver mudança em um sistema aplicado ao mundo real, pois, senão, o mesmo se tronará progressivamente menos útil.
Complexidade Crescente	Com as mudanças contínuas, um sistema tende a ser mais complexo, desta forma, deve-se alocar recursos para a refatoração do sistema de forma contínua para a sua simplificação.
Autorregulação	A evolução de programas está sujeita a uma autorregulação, tornando as medidas globais de esforço estatisticamente previsíveis. Portanto, números como tamanho, quantidade de erros (*bugs*) e tempo de lançamento (*release*) tendem a ser consistentes e relativamente constantes.
Estabilidade organizacional	Ao longo do ciclo de vida a taxa efetiva de trabalho é invariante no tempo, independente dos recursos destinados, contudo, as solicitações de usuário podem influenciar nesta estabilidade.

Fonte: Adaptado de Lehman e Belady (1985)

Quadro 1 – Leis de Lehman

(conclusão)

Conservação da familiaridade	Durante o ciclo de vida, a mudança incremental é aproximadamente constante, isso se dá, pois o sistema deve crescer conforme os indivíduos envolvidos conseguem absorver as novidades coletiva e individualmente.
Crescimento contínuo	Tende-se a aumentar continuamente as funcionalidades para manter a satisfação dos usuários.
Declínio de qualidade	A qualidade de um sistema sempre cairá, exceto se houver esforços destinados para que as mudanças do ambiente operacional sejam refletidas.
Sistema de *feedback*	Os processos de evolução incorporam sistemas de *feedback*, *multiloop* e multiagente, devendo tratá-los como sistemas de *feedback* para alcançar significativa melhoria do produto.

Fonte: Adaptado de Lehman e Belady (1985)

As Leis de Lehman fizeram avanços significativos na evolução do *software* em um sentido holístico e ainda são objeto de estudos experimentais sobre a evolução do *software*, que testam a validade atual das leis (GEZICI; TARHAN; CHOUSEINOGLOU, 2019).

As abordagens trazidas por Lehman e Belady (1985) e Sommerville (2011) destacam que as organizações devem considerar, ao se planejarem, o processo de manutenção, sempre se apoiando nas boas práticas de engenharia.

Com isso, se conclui que a manutenção de *software* é tão indispensável quanto qualquer outra fase do ciclo de vida de *software*, contudo ela conta com o complexo critério da imprevisibilidade. Caso a mesma se apoie nas boas práticas de engenharia, é possível que o *software* possua uma vida longa e duradoura, garantindo sua aplicabilidade. Não somente isso, avaliações para melhoria desta fase são essenciais para compreender o seu comportamento e reduzir custos envolvidos nestes processos, otimizando também o tempo e os recursos desprendidos a ela. Desta forma, ferramentas que auxiliem neste processo podem apoiar as decisões a serem tomadas pela área de manutenção de *software*.

2.2.1 Natureza da Manutenção

Diferente dos demais estágios de desenvolvimento, a manutenção de *software* não deve somente considerar o produto do desenvolvimento, mas sim se preocupar com o *software* atual,

garantindo que haja uma relação deste com seus usuários e operadores, avaliando com isso a satisfação com o sistema (PFLEEGER, 2004).

Corroborando com isso, Bourque e Fairley (2014) é taxativo ao afirmar que a manutenção é necessária para garantir que o *software* continua a satisfazer os requisitos do usuário, sendo aplicável a qualquer *software* desenvolvido em qualquer modelo de ciclo de vida, pois o *software* muda devia a ações corretivas ou não corretivas.

O autor Pfleeger (2004) aponta que a manutenção atua de forma simultânea em quatro aspectos principais da evolução do sistema:

- Manter o controle sobre as funções do dia-a-dia do sistema;
- Manter o controle sobre as modificações do sistema;
- Aperfeiçoar as funções aceitáveis já existentes;
- Tomar medidas preventivas para que o desempenho do sistema não diminua para níveis inaceitáveis.

Além de corroborar com os estes aspectos, Bourque e Fairley (2014) acrescenta que outro aspecto importante da manutenção é identificar ameças de segurança e corrigir vulnerabilidades de segurança.

Desta forma, nem todas as atividades da manutenção de *software* possuem a mesma natureza e podem ser caracterizadas da mesma forma. Para isso, Sommerville (2011) e Levin e Yehudai (2019) indicam que existem três diferentes tipos de manutenção de *software*: Correção de Defeitos, Adaptação Ambiental e Adição de Funcionalidade. E, ainda, Bourque e Fairley (2014) destaca que o IEEE 14764:2006 incluiu uma quarta categoria a preventiva. Estas categorias são melhor detalhadas no Quadro 2.

Quadro 2 – Tipos de Manutenção

(continua)

Correção de defeitos	Conhecida também como corretiva, trata-se de modificações reativas de um *software* para a correção de algum problema.
Adaptação Ambiental	Conhecida também como adaptativa, este tipo de manutenção ocorre quando algum aspecto ambiental (hardware, negócio, plataforma, etc.) sofre alguma mudança.

Fonte: Adaptado de Sommerville (2011) e Bourque e Fairley (2014).

Quadro 2 – Tipos de Manutenção

(conclusão)

Adição de funcionalidades	Conhecida também como perfectiva, trata-se da mudança dos requisitos do sistema em resposta a mudanças organizacionais ou de negócio. Trata-se de uma mudança para melhorar algum aspecto do sistema
Preventiva	Trata-se da manutenção para prevenir possíveis falhas do sistema antes de suas detecções por parte da operação.

Fonte: Adaptado de Sommerville (2011) e Bourque e Fairley (2014).

Embora, na prática, não haja grande distinção entre esses quatro tipos de manutenção e as mesmas podem acabar se confundindo, onde, por exemplo, ao adaptar o *software* a um novo ambiente, acaba-se por adicionar uma nova funcionalidade ou tira-se proveito para corrigir defeitos que possam ainda ocorrer (SOMMERVILLE, 2011). Contudo, cada atividade da manutenção de *software* pode ser categorizada de acordo com uma dessas categorias (LEVIN; YEHUDAI, 2019).

Outra categoria que pode ser trazida para as manutenções, apontada por Bourque e Fairley (2014), é a indicada pelo IEEE 14764:2006, que é demonstrada no Quadro 3.

Quadro 3 – Categorias de Manutenção de *software* (IEEE 14764:2006)

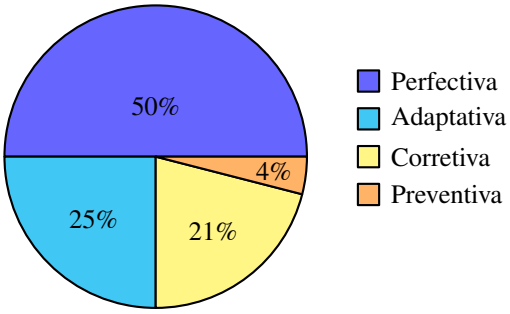
	Correção	**Aprimoramento**
Proativa	Preventiva	Perfectiva
Reativa	Corretiva	Adaptativa

Fonte: Adaptado de Sommerville (2011) e Bourque e Fairley (2014)

Percebe-se com isso que, corroborando com os aspectos da manutenção corretiva e adaptativa, as mesmas são reativas, ou sejam, aguardam o acontecimento de um fato para ocorrerem, já as manutenções preventivas e perfectivas devem ocorrer de maneira pró-ativa, sempre buscando a melhoria do *software*. Também devido as suas características, as manutenções corretivas e preventivas são de correção, pois buscam resolver falhas do *software*, já as manutenções perfectivas e adaptativas se tratam de aprimoramentos feitos no *software* de maneira a manter sua qualidade e utilidade.

Outro ponto importante no que tange as categorias de manutenção está no esforço implicado sobre elas. Sommerville (2011) aponta que a manutenção ocupa a maior fatia dos orçamentos destinados ao desenvolvimento de *software*, sendo que deste orçamento, conforme de destacada na Figura 3, a categoria de adição de funcionalidades é a que possui maior custo de investimento.

Figura 3 – Gastos com Manutenção por Categoria



Fonte: Adaptado de Pfleeger (2004).

Pode-se perceber que o custo envolvendo novas funcionalidades e aperfeiçoamento do sistema é muito maior do que o gasto com correções de problemas, por exemplo. Outro ponto interessante é que a categoria com menor gasto é a preventiva, isso se deve ao fato de que poucas organizações aplicam este tipo de manutenção, conforme destaca Pfleeger (2004), o que corrobora com sua baixa ocorrência.

Desta forma, Sommerville (2011) aponta que quanto maiores forem os esforços durante o desenvolvimento do sistema para produção de um sistema manutenível, menor serão os custos gerais durante a vida útil do sistema. Devido à redução potencial de custos de compreensão, análise e testes, existe um significativo efeito multiplicador quando o sistema é desenvolvido para manutenção.

Não somente isso, Nabilah e Sunindyo (2019) reforçam que a capacidade de manutenção do *software* está fortemente relacionado com o código dos programas desenvolvidos, pois um bom código pode ajudar os desenvolvedores a melhorar a qualidade da evolução do *software*, estando livres de erros.

Logo, é possível se evitar manutenções corretivas, tais como as corretivas e preventivas, desde que sejam empregados esforços pela manutenibilidade do sistema. Contudo, as manutenções adaptativas representam uma das principais despesas na área de manutenção, e aprimorar o gerenciamento dessas atividades pode resultar em uma redução nos custos desse investimento.

2.3 QUALIDADE DE *SOFTWARE*

A qualidade de *software* e a manutenção de *software* são áreas que andam em conjunto na engenharia de *software*. Sua desvinculação é complexa, pois, costumeiramente, as atividades de qualidade e manutenção tendem a ser necessariamente aplicadas em conjunto.

Alcançar a qualidade de *software* desejada torna-se difícil para os desenvolvedores, pois muitas vezes ultrapassam as restrições orçamentárias. Portanto, há necessidade de encontrar uma solução adequada que tenha a capacidade de minimizar os custos (TARWANI; CHUG, 2016).

Não somente isso, conforme traz Wazlawick (2013), a ISO/IEC 25010:2011, que determina o modelo de qualidade, define um conjunto características internas e externas de produto

de *software*, e dentre elas, está a capacidade de manutenção do *software*.

Contudo, é necessário primeiramente compreendermos do que se trata a qualidade de *software*. Conforme Pressman (2011), uma gestão de qualidade efetiva aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. Nesta mesma abordagem, o autor determina que a definição serviria para enfatizar três pontos:

- **Uma gestão de qualidade efetiva:** Aponta a importância das atividades de suporte da engenharia de *software* que tendem a fornecer todos os subsídios necessários para um desenvolvimento de *software* completo, pautado na manutenibilidade e qualidade do *software*;
- **Um produto útil:** Visa o fornecimento de um produto que atenda as necessidades e desejos do usuário final, com confiabilidade e isenção de erros, satisfazendo um conjunto de requisitos;
- **Ao agregar valor tanto para fabricante quanto para o usuário:** Determina que um produto de *software* deve gerar benefícios para a empresa de *software* bem como para a comunidade de usuários finais.

Hoje a garantia da qualidade do *software* é uma grande preocupação, tanto para os usuários finais, que desejam fazer seu trabalho da melhor forma possível, quanto para as empresas de desenvolvimento de *software*, que desejam oferecer a seus clientes serviços de alta qualidade (SIAVVAS; CHATZIDIMITRIOU; SYMEONIDIS, 2017).

Corroborando com a área de qualidade e manutenção, Sommerville (2011) destaca que os termos "garantia da qualidade"e "controle de qualidade"são amplamente utilizados na indústria manufatureira, onde a garantia de qualidade, ou QA (*quality assurance*), é a definição de processos e padrões que devem conduzir a produtos de alta qualidade e a introdução de processos de qualidade na fabricação, onde através do controle de qualidade, estes processos são aplicados para eliminar os produtos que não atingiram o nível de qualidade exigido.

A capacidade de manutenção é uma característica interna do *software*, conforme destaca Wazlawick (2013), determinada como uma das características chaves do modelo de qualidade de *software*. Ela pode ser percebida diretamente apenas pelos desenvolvedores, contudo os clientes podem ser afetados diretamente por ela. Trata-se da medição da facilidade em que se realizam alterações no *software*, seja para detectar e corrigir erros ou para sua evolução.

A definição trazida por Iqbal et al. (2021) corrobora com as citadas pelo IEEE (1983), determinando que a capacidade de manutenção de um *software* é a facilidade com que um sistema ou componente de *software* pode ser modificado.

Os autores Wazlawick (2013) e Molnar e Motogna (2020) ainda apontam que a capacidade de manutenção se subdivide em 5 (cinco) sub-características:

- **Modularidade:** Grau em que o sistema é logicamente subdividido, de forma que a manutenção de uma parte tenha o mínimo de impacto em outra;
- **Reusabilidade:** Grau em que as partes podem ser reaproveitadas;
- **Analisabilidade:** Permite encontrar (depurar) defeitos de forma fácil;
- **Modificabilidade:** Facilidade que o sistema oferece para correção de erros detectados sem serem introduzidos novos defeitos ou haver degradação da aplicação;
- **Testabilidade:** Facilidade na aplicação e realização de testes.

Outro ponto que merece destaque é que a qualidade de um determinado *software* não está ligado somente ao produto de *software*, conforme afirma Sommerville (2011), mas também ao processo de desenvolvimento utilizado em sua fabricação e as pessoas nos quais estão envolvidas nestes processos. A relação entre a qualidade de processo e de produto é mais complexa, o desenvolvimento de *software* é um processo criativo, em vez de mecânico, portanto, a influência de competências e experiências individuais é significativa, bem como os fatores externos (tais como a novidade de uma aplicação ou pressão comercial para um *release* anterior do produto) também afetam a qualidade de produto, independentemente do processo usado (SOMMERVILLE, 2011).

Reforçando essas questões, Pfleeger (2004) destaca como principais fatores pessoais, o entendimento limitado das necessidades dos usuários e as prioridades de gerenciamento que frequentemente se sobrepõem às prioridades técnicas e/ou são replanejadas. Por outro lado, os fatores técnicos incluem desafios como a conciliação entre a implementação de novas funcionalidades e a manutenção das existentes, a gestão de recursos e paradigmas aplicados, bem como as dificuldades na realização de testes.

Desta forma, a qualidade de *software* e manutenção de *software* estão ligadas diretamente, pois, para haver uma efetiva manutenção, é necessário que se haja uma efetiva qualidade no desenvolvimento de *software*. Desta forma, Wazlawick (2013) destaca que para haver esta gestão efetiva, é necessário que a mesma seja medida, a fim de determinar, com isso, seu real controle de qualidade.

2.4 MEDIÇÕES DA ÁREA DE *SOFTWARE*

A existência do processo de *software* sem haver medição traz instabilidade para o mesmo, haja visto que a estabilidade é somente alcançada a partir do momento em que seu desempenho futuro for previsível em limites estatisticamente estabelecidos (DEMING, 2000). As métricas de *software* fornecem avaliação quantitativa da qualidade do *software*, permitindo que os engenheiros alterem o curso do processo para promover a melhoria da qualidade do produto final (BIGONHA et al., 2019).

Reafirmando esse aspecto, Wazlawick (2013) indica que um processo de gerência, para ser mais eficaz, precisa se basear em medições, do contrário, não há como saber se a tarefa não está sendo feita ou se sua a qualidade é aceitável. Além disso, as métricas passam a desempenhar um papel crucial para melhorar o desenvolvimento da qualidade de *software*, logo, selecionar as métricas corretas também é de primordial importância para um desenvolvimento de *software* bem-sucedido (DAHAB et al., 2018).

Os autores Fenton e Pfleeger (2004) definem que medições são processos que atribuem valor a certas propriedades, de maneira a poder descrevê-las. Com isso, pode-se definir que na engenharia de *software*, essas propriedades, também conhecidas como perspectivas, são objetos ou eventos, que podem ser classificados em produtos, processos e recursos. Pressman (2011) também aponta que medição seria a coleta de medidas, sejam elas do processo, do produto ou dos recursos, nos quais auxiliam a engenharia de *software* a realizar devidamente a entrega do produto final.

Ainda, há dois tipos de atributos distintos, as internas, sendo aqueles passíveis de levantamento na própria organização (empresa), sem a necessidade de interferências de ambientes externos, e os externos, que são aquelas mensuráveis apenas com fatores externos, desta forma são dependentes do que está sendo medido e de onde será aplicado, conforme afirma Fenton e Pfleeger (2004) e Huda et al. (2018).

As medições dos atributos externos são complexas de serem realizadas e medidos objetivamente, conforme determina Sommerville (2011), logo, para extrair estas medições, por vezes se faz necessária a interpretação através de atributos internos relacionados a atributos externos, de maneira a inferir seu comportamento.

Contudo, Kitchenham (1990) destaca que apenas é possível realizar o relacionamento das métricas externas e internas se três fatores forem atendidos:

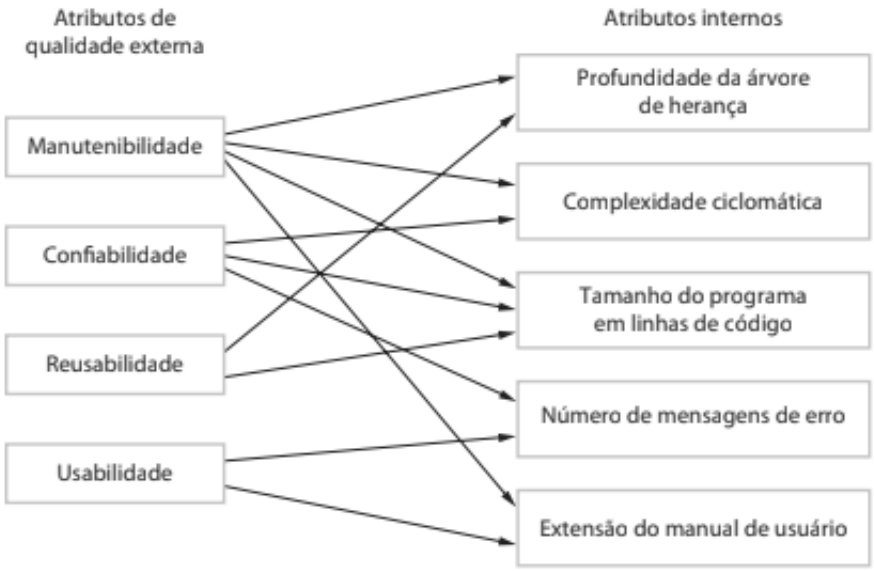
- **Precisão:** A medição dos atributos internos deve ocorrer com precisão, ou seja, deve refletir com exatidão o que está sendo medido ou estimado.
- **Relação:** A relação entre o atributo externo e interno deve existir e ser clara.
- **Fórmula:** A relação entre o atributo externo e interno deve ser possível de se expressar em uma fórmula ou modelo.

Sommerville (2011) cita e complementa que alguns atributos internos e externos podem ser relacionados intuitivamente, conforme destacado na Figura 4. Entretanto, mesmo que possa haver o relacionamento entre esses atributos, o diagrama não determina como eles acontecem.

Também é importante mencionar que Roche (1994) sugeriu as 5 (cinco) principais atividades envolvendo um processo de medição, sendo elas:

- **Formulação:** Criação de métricas apropriadas ao *software* a ser mensurado.
- **Coleção:** Mecanismo a ser utilizado para agrupar as métricas formuladas.

Figura 4 – Relação entre atributos internos e externos



Fonte: Adaptado de Sommerville (2011)

- **Análise:** Aplicação das métricas e ferramentas necessárias;
- **Interpretação:** Avaliação das métricas;
- **Feedback:** Recomendações extraídas sobre os resultados.

Logo, somente serão úteis as métricas que passem por estas atividades, pois, com isso, é possível determinar que as métricas atingiram os seus objetivos.

Pressman (2011) cita e complementa que as métricas de *software* serão úteis somente se forem efetivamente caracterizadas e validadas de forma que demonstrem valer a pena, ou seja, que sejam úteis para a avaliação das perspectivas. Diante disso, o autor elabora alguns princípios que podem servir como caracterização e validação das métricas:

- **Propriedade Matemática:** O valor da métrica deverá estar em um intervalo significativo e possuir uma escala racional.
- **Relacional com as Características do *software*:** Quando uma métrica representa uma característica do *software* que aumenta em ocorrências positivas ou diminui em ocorrências indesejadas, o valor da métrica deverá aumentar ou diminuir da mesma maneira.
- **Validação empírica:** Cada métrica deverá ser validada empiricamente em uma ampla variedade de contextos antes de ser publicada ou usada para tomada de decisões.

Para complementar, os autores Pressman (2011) e Roche (1994) afirmam que para uma efetiva coleta e análise dos resultados é necessário que, sempre que possível, sejam feitas de forma automática, que sejam aplicadas técnicas estatísticas válidas para estabelecer relações

e que diretrizes, e recomendações interpretativas, sejam estabelecidas para cada métrica. Não somente isso, Huda et al. (2018) afirma que esta automação é essencial para simplificar o monitoramento, medição, análise e rastreamento de todas as métricas das diferentes perspectivas do *software*.

Por fim, pode-se citar que as medições podem ser subdividas em variáveis independentes, dependentes e métricas, conforme mencionam García et al. (2006), explicados a seguir, com base na ISO/IEC 15939:2007 (ISO/IEC, 2007):

- **Variáveis Independentes (*Base Measures*):** Medida definida por um único termo e seu método para qualificá-lo. Trata-se de uma medida funcionalmente independente, que captura informações sobre um único atributo.
- **Variáveis Dependentes (*Derived Measures*):** Medida definida pela função de dois ou mais valores das variáveis independentes. Podem capturar as informações de dois ou mais atributos, ou o mesmo atributo de várias entidades.
- **Métricas (*metrics*):** Medida que fornece uma estimativa ou avaliação de atributos especificados. Servem de base para a tomada de decisões, de maneira a quantificar ou precisar valores que por si só podem ser imperfeitos, ou seja, que sozinhos muitas vezes não trazem todas as informações necessárias.

Logo, se pode destacar que as medições de *software* são imprescindíveis para determinar a qualidade de um determinado *software* e que sem elas, não seria possível gerenciá-lo corretamente. Ainda, é necessário destacar a importância da categorização destas medições, pois, através delas, é possível determinar as relações entre as métricas internas e externas, permitindo que métricas com mensurações não objetivas sejam avaliadas.

2.4.1 Medições da Manutenção de *software*

Não diferente das demais áreas de desenvolvimento de *software*, a área de manutenção de *software* pode se favorecer das medições para as suas tomadas de decisão, orientado as atividades, ajudando a avaliar o impacto de uma mudança ou a avaliar os méritos relativos as diversas modificações, ou abordagens propostas (PFLEEGER, 2004).

Além disso, como aborda Sommerville (2011), gerentes não costumam gostar de surpresas, principalmente quando há um custo elevado com isso, desta forma, as métricas auxiliam nesta tomada de decisão, para determinar quais mudanças no sistema podem ser propostas e que partes do sistema são, provavelmente, as mais difíceis de serem mantidas.

Avaliando os atributos externos da manutenção de *software*, que são destacados por Pfleeger (2004), o tempo é um dos principais fatores que afetam a manutenção, seja ele em análise dos problemas relatados ou em realização das mudanças propostas, contudo não somente ele deve ser levado em consideração, mas também a dependência do caráter pessoal, haja visto

que são pessoas que realizam a manutenção, a documentação e o ambiente onde o *software* está inserido.

A avaliação dos atributos externos, por não serem objetivos, depende diretamente de uma métrica interna precisa. Com isso, muitos pesquisadores propuseram medidas para atributos internos relacionados à manutenibilidade (PFLEEGER, 2004; MOLNAR; NEAMTU; MOTOGNA, 2020).

Uma das principais constatações que se chegou com relação a estes estudos, conforme menciona Sommerville (2011), é de que quanto mais complexo o sistema, mais cara será a manutenção, o que traz uma conexão nítida entre produtos mal estruturados e mal documentados e sua manutenibilidade (PFLEEGER, 2004; BOEHM, 2006).

Diante disso, uma das principais métricas para a compreensão da complexidade de código-fonte é o número ciclomático, sendo uma medida que considera um aspecto da complexidade estrutural do código-fonte, medindo o número de caminhos linearmente independentes ao longo do código (PFLEEGER, 2004).

Desta forma, a complexidade ciclomática é uma métrica de *software* que fornece uma medida quantitativa da complexidade lógica de um programa, tendo fundamento na teoria dos grafos e fornece uma métrica de *software* extremamente útil, aplicada tanto para avaliação da completude de validação e testes de *software*, quanto para determinar o quão complexo é a manutenção de um código-fonte (PRESSMAN, 2011; HUDA et al., 2018).

Com isso, pode-se assumir que programas com complexidade ciclomática menor ou igual a 10 são simples e fáceis de manutenção, com complexidade ciclomática de 11 a 20 são de médio risco de manutenção, de 21 a 50 são de alto risco e acima de 50 não são manuteníveis, sendo necessária sua reescrita (WAZLAWICK, 2013; FRANTZ et al., 2019).

Algumas outras abordagens utilizadas para calcular o esforço de manutenção são a equação de Belady-Lehmaan, COCOMO II e Legibilidade de Código, no qual são aplicados cálculos estatísticos para determinar o quão complexa será a manutenção e qual será o seu custo, conforme destaca Pfleeger (2004). Estas métricas serão melhor abordadas e destacadas no Capítulo 3.

Bourque e Fairley (2014) sugere que as medidas de manutenção de *software* devem avaliar quatro sub-caracterísitcas, sendo elas a analisabilidade, que é a capacidade de diagnosticar uma falha, a mutabilidade, que deve avaliar o esforço associado a implementação e/ou modificação, a estabilidade, que avalia o comportamento inesperado do *software*, e a testabilidade que determina o esforço para tentar validar a alteração realizada. Também são medidas importantes o tamanho do *software*, a complexidade, a compreensibilidade e a manutenibilidade.

Entretanto, estas abordagens apenas consideram como métricas internas as avaliações dos produtos de *software*, como o seu código-fonte, e pouco exploram as demais perspectivas, que serão apontadas na seção 2.4.2, conforme destaca Sommerville (2011), sendo que a maioria dos gerentes apenas combina as métricas de produto com a intuição e a experiência para estimar os custos e esforço.

2.4.2 Perspectivas das Medições

Existem várias medidas de *software* que podem ser derivadas dos atributos do *software*, do processo de manutenção e do pessoal, incluindo tamanho, complexidade, qualidade, compreensibilidade, manutenibilidade e esforço (BOURQUE; FAIRLEY, 2014).

Contudo, as medições da manutenção de *software* ocorrem dentro de três perspectivas distintas, sendo elas (APRIL; ABRAN, 2008; IQBAL et al., 2021):

- **Recurso:** São utilizados para a realização da manutenção, tais como Pessoas, Ferramentas, Ambiente e Orçamento.
- **Processo:** São as etapas utilizadas para que a manutenção ocorra, tais como Suporte operacional, Correções, Evoluções, Monitoramento de mudanças e Teste de Regressão.
- **Produto:** São as entregas capazes de suprir a necessidade do *software*, tais como Documentações Técnicas e de Usuário, Códigos Fontes, Bancos de Dados e Relatório de Análise de Impacto.

Não distante disso, Bourque e Fairley (2014) determina que as perspectivas ligadas à manutenção de *software*, cujos atributos podem ser submetidos à medição são destacadas como produto, processo e recurso.

É importante destacar que a evolução de *software* traz consigo a evolução e modificação dessas perspectivas (VARGA, 2018), o que se relaciona com a dinamicidade da área de manutenção de *software*. Além disso, é necessário destacar o ambiente de estudo e o objeto estudado, denominado de contexto por Petersen e Wohlin (2009), avaliando os estudos conforme sua aplicabilidade.

Logo, apenas realizando as medições é que se pode aprender mais sobre o comportamento dos recursos, dos processos e do produto de *software* e suas relações de causa e efeito, pois os recursos da área de manutenção seguem os processos da área de manutenção que, por consequência, produzem os produtos da área de manutenção (APRIL; ABRAN, 2008).

2.4.3 Complexidade das Medições

Todos os problemas relativos à manutenção de um sistema, contribuem para o alto custo da manutenção de *software* Pfleeger (2004). A equipe de manutenção realiza sempre uma tarefa dupla, primeiro entendendo o projeto, código, filosofia e estrutura do sistema e, posteriormente, desenvolvendo o necessário para atender os requisitos de sua tarefa (LEVIN; YEHUDAI, 2019).

Além disso, devido as constantes mudanças de projeto, prioridades e estimativas, sempre cabe a equipe de manutenção equilibrar os objetivos de curto e longo prazo, decidindo quando deverá ser reduzida a qualidade em prol de aumentar a velocidade de entrega (PFLEEGER, 2004).

Ainda, as atividades de manutenção não são constantes ou similares, conforme destacam Wazlawick (2013) e Jones (1998), mas sim variadas e que podem ser categorizadas da seguinte forma:

- **Reparação de Defeitos:** Se trata da atividade base da manutenção, responsável em eliminar problemas que não deveriam existir;
- **Remoção de Módulos Falhos:** Defeitos tentem a se concentrar em determinados módulos, logo sua remoção e reestruturação é necessária;
- **Suporte a Usuários:** Interface entre o cliente e a empresa do *software*, responsável por receber reclamações, realizar a triagem e encaminhar soluções previamente conhecidas;
- **Migração entre Plataformas:** Trata-se da conversão da base de desenvolvimento de um sistema para outra;
- **Conversão de Arquitetura:** Trata-se da conversão da tecnologia do *software*, como banco de dados, linguagem de desenvolvimento, entre outros;
- **Adaptações Obrigatórias:** Alterações obrigatórias decorrentes de normativas ou leis;
- **Otimitização de Performance:** Otimizar a performance resolvendo gargalos da aplicação, normalmente relacionados a dados, processamento e números de usuários simultâneos;
- **Melhorias:** Originados costumeiramente por solicitação dos clientes para a introdução de novas funcionalidades.

Conforme mencionado por Ghanem (2020), além dessas categorias propostas, deve ser feita uma análise empírica das atividades exercidas pela organização, para identificar novas categorias.

Desta forma, estimar e medir as atividades da manutenção de *software*, em geral, se trata de uma situação complexa, pois, além de envolver uma análise múltipla de perspectivas, é necessário considerar os diferentes processos que podem ocorrer no ambiente de manutenção, bem como também a dinamicidade da área de manutenção de *software*.

Baseado nos conceitos de manutenção e métricas da engenharia de *software*, para auxiliar no desenvolvimento do *framework* proposto, foi utilizada a abordagem de *Design Science Research Methodology* para estruturar a produção de um artefato de valor científico, tratada na seção a seguir.

2.5 *DESIGN SCIENCE RESEARCH METHODOLOGY*

O *Design Science Research* é uma metodologia cujo objetivo é legitimar o desenvolvimento de artefatos como um meio para a produção de conhecimentos científicos do ponto de vista epistemológico (PIMENTEL; FILIPPO; SANTORO, 2020).

Os artefatos não se restringem somente aos objetos físicos, mas sim também são considerados como algo projetado, um engenho, uma artificialidade, logo, as abstrações também são artefatos, pois qualquer coisa projetada para alcançar um objetivo pode ser considerado um artefato (PEFFERS et al., 2007; PIMENTEL; FILIPPO; SANTORO, 2020). Desta forma, os *frameworks* são artefatos que servem como suporte ou guia na realização de alguma determinada atividade (PIMENTEL; FILIPPO; SANTORO, 2020).

Além disso, esta metodologia envolve um processo rigoroso para projetar artefatos para resolver problemas observados, fazer contribuições de pesquisa, avaliar os projetos e comunicar os resultados ao público apropriado (PEFFERS et al., 2007).

Logo, Wieringa (2014) afirma que a ciência do design é o design e a investigação de artefatos em um determinado contexto, onde estes artefatos são projetados para interagir com este determinado problema, para melhorar algo nesse contexto.

Considerando isso, a criação de artefatos para abordar um problema específico, sem a necessidade de buscar uma solução ótima, mas sim uma solução satisfatória para a situação, é reconhecida como uma forma válida de produção de conhecimento científico, desde que sejam seguidos os procedimentos metodológicos apropriados, como apontado por Dresch, Lacerda e Antunes (2015).

São apresentadas nas próximas seções, as motivações e aplicações do *Desing Science Research Methodology* nesta pesquisa para legitimar o artefato produzido.

2.5.1 Aplicação

Para que o processo metodológico seja aplicado corretamente, Hevner et al. (2004) mencionam que a pesquisa deve possuir algumas diretrizes que devem ser seguidas, conforme demonstra o Quadro 4, estas diretrizes tem o propósito de auxiliar os pesquisadores, revisores, editores e leitores para entender os requisitos para uma pesquisa científica de design eficaz. Não se tratam de diretrizes obrigatórias ou rotineiras, pois os pesquisadores, revisores e editores devem usar suas habilidades criativas e julgamento para determinar quando, onde e como aplicar cada uma das diretrizes em um projeto de pesquisa específico, porém estas devem ser abordadas de alguma forma para que o *Design Science Research* seja considerada concluída.

Quadro 4 – Diretrizes do *Design Science Research*

Design como um Artefato	A pesquisa deve produzir um artefato viável na forma de uma construção, um modelo, um método ou uma instanciação.
Relevância do problema	O objetivo da pesquisa é desenvolver soluções baseadas em tecnologia para problemas de negócios importantes e relevantes.
Avaliação do Projeto	A utilidade, qualidade e eficácia de um artefato de design devem ser rigorosamente demonstradas por meio de métodos de avaliação bem executados.
Contribuições da Pesquisa	A pesquisa científica de design eficaz deve fornecer contribuições claras e verificáveis nas áreas do artefato de design, fundamentos de design e/ou metodologias de design.
Rigor da Pesquisa	A pesquisa depende da aplicação de métodos rigorosos tanto na construção quanto na avaliação do artefato de design.
Design como um processo de pesquisa	A busca por um artefato eficaz requer a utilização dos meios disponíveis para alcançar os fins desejados, ao mesmo tempo em que satisfaz as leis do ambiente do problema.
Comunicação da Pesquisa	A pesquisa deve ser apresentada eficazmente tanto para o público orientado para a tecnologia quanto para o público orientado para a gestão.

Fonte: Adaptado de Hevner et al. (2004)

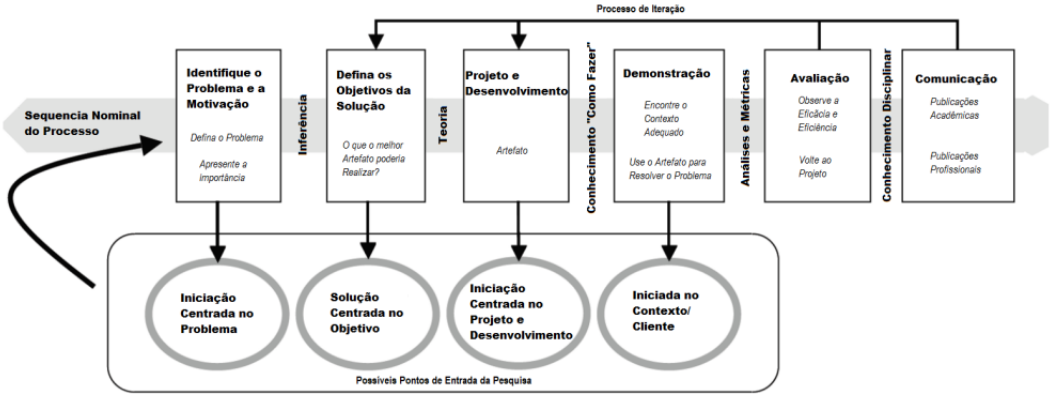
Além disso, Pimentel, Filippo e Santoro (2020) afirmam que a adoção de um método estabelecido para conduzir pesquisas no contexto da abordagem do *Design Science Research* é benéfica para auxiliar os pesquisadores a alcançar o rigor metodológico. Não existe uma metodologia única, mas sim diversas propostas para organizar as etapas de uma pesquisa, alinhadas com as diretrizes do *Design Science Research*. Com base nas evidências quantitativas apresentadas por Pimentel, Filippo e Santoro (2020), o método de pesquisa proposto por Peffers et al. (2007), intitulado *Design Science Research Methodology*, é amplamente empregado e possui um alto número de citações.

Além disso, esse método enfatiza a criação de artefatos práticos como resultado da pesquisa, combinando princípios de design para desenvolver artefatos que sejam tanto úteis quanto eficazes. Ele também mantém a rigidez e validação inerentes à pesquisa científica, proporcionando uma abordagem abrangente e integrada. Com base nessas considerações, optou-se por utilizar esse método para conduzir a presente pesquisa.

2.5.2 Etapas

De acordo com Peffers et al. (2007), existem seis etapas específicas para a aplicação do *Design Science Research Methodology*, sendo estas a Identificação do Problema e Motivação, Definição dos Objetivos da Solução, Projeto e Desenvolvimento, Demonstração, Avaliação e Comunicação, conforme destacado na Figura 5.

Figura 5 – Proceso do *Design Science Research Methodology*



Fonte: Pimentel, Filippo e Santoro (2020)

O detalhamento destas etapas, conforme as diretrizes de Peffers et al. (2007), o seguinte:

- **Identificação do Problema e Motivação:** Deve ser definido o problema de pesquisa específico, com a finalidade de identificar qual problema o artefato estará solucionando, bem como justificar o valor da solução, engajando tanto o pesquisador quanto o público para a busca da solução e aceitação de seu resultado;
- **Definição dos Objetivos da Solução:** Deve ser inferido os objetivos da solução a partir da definição do problema, sendo que estes objetivos podem ser quantitativos, quando já há uma solução e se deseja resultados melhores, ou qualitativos, quando se espera que o artefato forneça solução a problemas ainda não resolvidos;
- **Projeto e Desenvolvimento:** Esta atividade inclui determinar a funcionalidade desejada do artefato e sua arquitetura e, em seguida sua real criação;
- **Demonstração:** A demonstração do uso do artefato na solução de uma ou mais instâncias do problema pode envolver a sua experimentação, simulação, estudo de caso, prova ou qualquer outra atividade apropriada;
- **Avaliação:** Trata-se da observação e mensuração do quão bem o artefato suporta a solução do problema, envolve comparar os objetivos com os resultados reais observados com o uso do artefato na demonstração;

- **Comunicação:** Comunicar o problema e sua importância, o artefato, sua utilidade e novidade, o rigor de seu design e sua eficácia para pesquisadores e outros públicos relevantes, como profissionais atuantes na área de engenharia de *software*, quando apropriado.

Este processo é iterativo, pois, ao atingir as etapas de avaliação e comunicação, podem ser identificadas melhorias significativas que tragam alterações para o objetivo, projeto e desenvolvimento do artefato elaborado, sendo que cabe aos pesquisadores decidirem pela iteração ao fim das respectivas etapas (PEFFERS et al., 2007).

Desta forma, ao final das etapas, é possível determinar que houve a resolução de um determinado problema através da construção de um artefato, que, por usa vez, devido a todo rigor metodológico aplicado, possui caráter científico e pode ser aceito na produção de conhecimento (PEFFERS et al., 2007).

2.5.3 Avaliação do artefato

De maneira a estruturar a avaliação do artefato, os autores optaram pela utilização do questionário TAM (*Techology Acceptance Model*), proposto por (DAVIS, 1989), por ser um instrumento utilizado para medir a aceitação de tecnologia pelos usuários.

Esse modelo se fundamenta em três construtos principais: “utilidade percebida”, “facilidade de uso percebida” e “intenção de uso”. O questionário tem como finalidade avaliar a percepção dos usuários em relação à utilidade da tecnologia, ou seja, como ela melhora o desempenho e eficácia na execução de tarefas, a facilidade percebida no uso dessa tecnologia, e qual a intenção dos usuários de adotar a ferramenta. O propósito é compreender como esses fatores influenciam a aceitação e adoção de uma tecnologia específica pelos usuários (DAVIS, 1989).

O modelo TAM é amplamente reconhecido e utilizado na área de pesquisa em aceitação de tecnologia, conforme apontado por (AGUIAR; KEMCZINSKI; GASPARINI, 2017). Ele se baseia em fundamentos teóricos sólidos, fornecendo uma estrutura conceitual bem estabelecida para compreender a adoção de tecnologia pelos usuários, sendo utilizado no meio acadêmico e profissional, bem como é aderente à avaliação de *software* para gestão de projetos.

2.6 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram abordados os principais conceitos relacionados a ciclo de vida de desenvolvimento de *software*, manutenção de *software* e sua natureza, qualidade de *software* e medições na área de *software* que possuem caráter fundamental para a definição do modelo do *framework* de análise da área de manutenção de *software* proposto neste trabalho.

É importante destacar que a área de manutenção de *software* tem uma enorme importância no ciclo de vida de desenvolvimento de *software*, que por vezes não é devidamente considerada pelas organizações. O custo com a manutenção de *software* tende a ser o mais elevado do

ciclo de vida e, seu processo, deve ser avaliado, estruturado e considerado deste o começo do desenvolvimento de *software*, de maneira a tentar reduzir o custo futuro por negligência.

Através das métricas de manutenção de *software*, é possível ter uma perspectiva e previsão da área de manutenção de *software*. Entretanto, uma análise completa da área, hoje, demanda um grande esforço, bem como as pesquisas pouco exploram as perspectivas de processo e recursos aplicáveis a manutenção de *software*. Logo, uma análise do conjunto de todas as perspectivas permitiria uma maior precisão com relação às estimativas, trazendo uma maior assertividade para os gerentes e auxiliando na tomada de decisão.

Desta forma, um *framework* que permita considerar a dinamicidade da área de manutenção, sua devida complexidade e suas perspectivas, auxilia os gestores, permitindo, com isso, trazer maior segurança na tomada de decisões para a área de manutenção de *software*. Logo, é de suma importância que este considere que, diferente das demais áreas da engenharia de *software*, a manutenção de *software* possui a característica da imprevisibilidade, pois as necessidades e características do *software* se modificam no decorrer do projeto.

Por fim, visando conferir legitimidade a um artefato como meio de produção de conhecimento científico, é imprescindível aderir a um rigor técnico na sua elaboração. Nesse contexto, a aplicação do DSRM (*Design Science Research Methodology*) valida o artefato, começando pela identificação do problema e definição de objetivos. O framework é concebido e desenvolvido seguindo princípios bem definidos, com demonstrações práticas. Sua eficácia é avaliada por meio de métricas e estudos de caso, sendo os resultados comunicados, garantindo assim sua validade.

Com base na fundamentação teórica descrita neste capítulo, no Capítulo 3 são apresentados os trabalhos relacionados, que fazem um levantamento das principais métricas abordadas pela literatura e na indústria, permitindo identificar quais conjuntos de métricas são aplicáveis.

3 TRABALHOS RELACIONADOS

Neste capítulo são descritas a Revisão Sistemática da Literatura (RSL), que serviu como base para o levantamento das métricas da área de manutenção de *software*, e a pesquisa exploratória, cuja proposta foi realizar uma correlação do que foi identificado na revisão e o que é aplicado na indústria, em específico nas empresas de desenvolvimento de *software* da região de Joinville.

A RSL, descrita em Machado, Kemczinski e Schroeder (2023), teve o objetivo de identificar as principais métricas apontadas na literatura utilizadas na área de manutenção de *software*, com base em palavras-chave e nos critérios de inclusão e exclusão definidos, foram selecionados 44 artigos de um total de 2145. Assim, as métricas identificadas puderam ser categorizadas em perspectivas de produto, processo e recursos, destacadas na Seção 2.4.2. Além disso, foi observado que as validações da aplicação dessas métricas são realizadas por meio de experimentação técnica de medição, significando que as validações são conduzidas em ambientes controlados, o que muitas vezes não reflete com precisão a realidade do ambiente de manutenção de *software*.

Baseado na RSL, foi realizada uma pesquisa exploratória, por meio de um questionário aplicado com profissionais da área de manutenção de *software*, visando confrontar as métricas trazidas pela literatura, de maneira a compreender se são aplicadas no contexto real. Nesta pesquisa foi possível identificar que as métricas apontadas pela literatura são utilizadas pela indústria, porém os mesmos consideram que elas são pouco exploradas, sendo detalhada na Seção 3.2.

3.1 REVISÃO SISTEMÁTICA DA LITERATURA (RSL)

O objetivo da RSL de Machado, Kemczinski e Schroeder (2023) consistiu na identificação das métricas na área de manutenção de *software*, visando avaliar tanto os produtos de desenvolvimento de *software* quanto os recursos e os processos de desenvolvimento de *software*.

Neste estudo, foram seguidas as etapas propostas por Petersen et al. (2008). Inicialmente, foram formuladas as perguntas de pesquisa e identificados os critérios de inclusão (CI) e exclusão (CE). Em seguida, foram realizadas buscas por estudos relevantes, sendo realizada a extração e avaliação qualitativa desses estudos. Por fim, os dados foram sintetizados para responder às perguntas de pesquisa definidas e também para avaliar a validade das conclusões obtidas.

Desta forma, a Questão de Pesquisa (QP) e as seguintes Questões Secundárias (QS) levantadas podem ser observadas no Quadro 5:

Quadro 5 – Questões de Pesquisa

Questões de Pesquisa	**Motivação**
QP: Quais as variáveis utilizadas para a avaliação da área de manutenção de *software* e como são aplicadas?	Entender como ocorre a avaliação da área de manutenção de *software*.
QS1: Quais os principais indicadores para avaliação na área de manutenção de *software*?	Entender quais os indicadores utilizados para a área de manutenção de *software*.
QS2: Quais são as variáveis utilizadas pelos indicadores de manutenção de *software* e qual a sua ordem de aplicação?	Entender como essas variáveis afetam a avaliação da área de manutenção de *software*.
QS3: Quais os indicadores utilizados para avaliar e planejar as tarefas de manutenção de *software* e suas perspectivas de aplicação?	Entender como os indicadores influenciam no planejamento da área de manutenção de *software*.
QS4: Quais as ferramentas utilizadas para mensurar os indicadores na manutenção de *software*?	Entender as ferramentas hoje utilizadas pela área de manutenção de *software*.

Fonte: Adaptado de Machado, Kemczinski e Schroeder (2023)

Outra característica desta pesquisa foi a utilização do método PICO (População, Intervenção, Comparação e Resultados — *Outcomes*), sugerido por Kitchenham e Charters (2007) para a elaboração da *string* de busca, onde os componentes, definições e palavras-chave são destacadas no Quadro 6.

Quadro 6 – Estratégia PICO

Componente	**Definição**	**Palavras-Chave**
População	Manutenção de *Software*	"*software maintenance*"; "*support software*"; "*maintainability software*"
Intervenção	Métrica	"*indicator*"; "*measure*"; "*metric*"; "*quantitative*"; "*kpi*"
Comparação	Não aplicável	Não aplicável
Resultados	Identificação das métricas utilizadas na área de Manutenção de *Software*	Não aplicável

Fonte: Adaptado de Machado, Kemczinski e Schroeder (2023)

A *String* de busca final, após a aplicação da estratégia PICO foi a seguinte:

"software engineering"AND ("maintenance effort"OR "maintenance process"OR "maintainability") AND ("indicator" OR "measure" OR "metric" OR "quantitative" OR "kpi") AND NOT ("case study")

Ainda, a escolha dos Mecanismos de Busca (MBA) utilizados se deu com base no levantamento realizado por Buchinger, Cavalcanti e Hounsell (2014), que elencaram os MBAs que apresentam as melhores condições para a busca de conteúdos científicos para a área de Ciência da Computação e áreas afins.

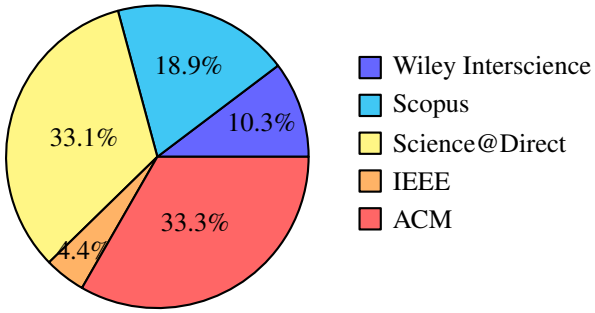
Destaca-se ainda que houve a aplicação de um procedimento de validação da qualidade dos artigos selecionados, sendo aplicado o padrão abordado por Huang et al. (2018). Para isso, foram aplicadas perguntas qualitativas, visando avaliar a credibilidade dos artigos, onde as possíveis respostas foram Sim, Parcial e Não, sendo pontuadas respectivamente com 1 ponto,

0,5 pontos e 0 pontos. Para aceitação dos artigos qualitativamente, os valores das pontuações em cada pergunta foram somados e apenas foram selecionados os estudos com somatório superior a 5,1 pontos, sendo considerados acima da média de respostas.

Por fim, para auxiliar a condução do estudo sistemático, foi utilizada a ferramenta **Parsifal**, disponível no endereço eletrônico (*https://parsif.al/*), permitindo os pesquisadores trabalharem de maneira colaborativa.

A pesquisa teve como data de término 25 de abril de 2020, apresentando um total de 2145 artigos, com a seleção final de 44 artigos analisados. Na Figura 6 são apresentados os resultados em percentual dos artigos aceitos por mecanismos de busca.

Figura 6 – Artigos identificados por MBA



Fonte: Machado, Kemczinski e Schroeder (2023)

Observa-se que os MBAs que mais trouxeram artigos foram ACM e Science@Direct, entretanto, nas análises de seleção destes artigos, poucos da Science@Direct foram efetivamente aproveitados. Já o MBA com melhor resultado foi a Scopus ao fornecer a maior quantidade de artigos selecionados.

Os dados extraídos dos MBAs selecionados foram classificados de maneira a responder às questões de pesquisa elaboradas e sua classificação pode ser observada no Quadro 7. Ainda, para classificar a forma de validação foram utilizados os mesmos métodos de validação definidos por Ejiogu (1993), sendo o exame teórico que avalia o atributo alvo do comportamento, a satisfação matemática da medição, a experimentação técnica da medição, o exame dos resultados e a confirmação da consistência com a análise dimensional.

Quadro 7 – Classificação dos Dados

(continua)

Classificação	**Descrição**
Ano de Publicação	Identificar a atualidade da métrica e sua evolução no passar do tempo.

Fonte: Elaborado pelo autor (2023)

Quadro 7 — Classificação dos Dados

(conclusão)

Métrica Proposta	Quantificar as métricas propostas de maneira a definir a quantidade de vezes que esta é proposta
Ferramenta Utilizada	Avaliar as ferramentas utilizadas pela área de Manutenção de *software* e suas relações com as métricas.
Perspectiva	Determinar se a métrica em questão avalia o Produto, o Processo ou o Recurso.
Categoria	Para cada perspectiva, as métricas foram distribuídas em categorias específicas.
Forma de Validação	Validar formalmente a métrica conforme preconiza Ejiogu (1993)

Fonte: Elaborado pelo autor (2023)

Durante a análise dos estudos selecionados, também foram padronizadas as nomenclaturas e definições das métricas, visando quantificar corretamente sua ocorrência. Além disso, quando as informações não estavam explicitadas durante a análise, estas foram inferidas com base na interpretação do conteúdo.

3.1.1 Principais Resultados

Machado, Kemczinski e Schroeder (2023) identificam que o número de publicações não é constante, mas ocorre recorrentemente. Além disso, observaram que o método de validação mais comumente utilizado para as métricas é a experimentação técnica de medição.

Com base nisso, conclui-se que as abordagens passam por períodos de experimentação antes de alcançar conclusões e conduzir novos estudos e avaliações. É importante notar que essa experimentação tende a ocorrer em um ambiente controlado.

Outra característica identificada é que as discussões sobre métricas da área de manutenção de *software* são trazidas para a socialização dos pesquisadores, pois os principais locais de publicações são conferências, demonstrando a característica evolutiva das pesquisas nesta área.

No que tange ao levantamento das métricas utilizadas na área de manutenção de *software*, afirma-se que a maioria das métricas propostas possuem correlação com a perspectiva de produto de *software*, e sofrem principalmente a influência de características internas da organização. Foram identificadas 300 métricas nas análises dos estudos, onde estas foram segregadas conforme a sua perspectiva (produto, processo e recurso) [1].

As 10 (dez) principais métricas ligadas a produto podem ser identificadas no Quadro 8.

[1] Produto, processo e recurso são as classificações da engenharia de *software* e da manutenção de *software* dada as perspectivas que podem ser medidas.

Quadro 8 – As dez principais métricas identificadas para a perspectiva de produto

Métrica	**Descrição**	**Quantidade**
LOC	Linhas de Código	21
WMC	Método Ponderado por Classe	20
DIT	Profundidade na Árvore de Herança	17
CBO	Acoplamento entre classes	16
RFC	Respostas para a Classe	16
NOC	Número de Filhos	15
LCOM	Falta de Coesão de Métodos	14
N	Tamanho do Programa	10
CC	Complexidade da Classe	8
NOA	Número de Atributos	7

Fonte: Machado, Kemczinski e Schroeder (2023)

Portanto, as principais métricas de produto estão associadas ao tamanho, complexidade e dificuldade de compreensão da codificação. Isso indica que produtos com menor tamanho e complexidade tendem a receber avaliações mais positivas e necessitam de menos intervenções de manutenção.

Ainda, as 10 (dez) principais métricas ligadas a processo podem ser identificadas no Quadro 9.

Quadro 9 – As dez principais métricas identificadas para a perspectiva de processo

Métrica	**Descrição**	**Quantidade**
MI	Indice de Manutenibilidade	3
NPR	Número de funções que participam do processo	3
NDA	Número de dependências de precedência entre atividades	3
NDWP	Número de dependências entre produtos de trabalho e atividades	3
NDWPIn	Número de dependências de entrada dos produtos de trabalho com as atividades no processo	3
NDWPOut	Número de dependências de saída dos produtos de trabalho com as atividades no processo	3
ASP	Pontos de Historia Ajustados (*Adjusted Story Points*)	3
DOC	Atualização de Documentação	2
TST	Testabilidade (*Testability*)	2
SES	Conformidade com os padrões de engenharia de *software*	2

Fonte: Machado, Kemczinski e Schroeder (2023)

Cabe ressaltar que estas métricas estão relacionadas principalmente a quantidade de etapas a serem executadas em um determinado processo, levando ao fato de que quanto mais etapas e/ou dependências durante o processo, maior será o tempo de execução e atendimento de uma atividade de manutenção de *software*.

Por fim, as 10 (dez) principais métricas ligadas a recurso podem ser identificadas no Quadro 10.

Quadro 10 – As dez principais métricas identificadas para a perspectiva de recurso

Métrica	**Descrição**	**Quantidade**
E	Esforço de Trabalho	5
D	Dificuldade de Desenvolvimento	2
BDD	Complexidade de domínio do sistema	1
CONF	Confiança na resolução de tarefas	1
DEVMSYS	Experiência de desenvolvimento de módulo	1
DEVSYS	Experiência total de desenvolvimento	1
ESS	Experiência com *software*	1
FPL	Familiaridade com a linguagem de desenvolvimento	1
MEXPAPP	Experiência de manutenção na aplicação	1
MEXPTOT	Experiência total de manutenção	1

Fonte: Machado, Kemczinski e Schroeder (2023)

Os fatores humanos são relacionados diretamente as métricas de recurso, compreendendo que identificar o envolvimento humano e melhorar seu engajamento são fatores que beneficiam a área. Além disso, a experiência prática ou teórica, também influencia diretamente, pois 6 (seis) das principais métricas são relacionadas ao nível de experiências que estes recursos humanos possuem.

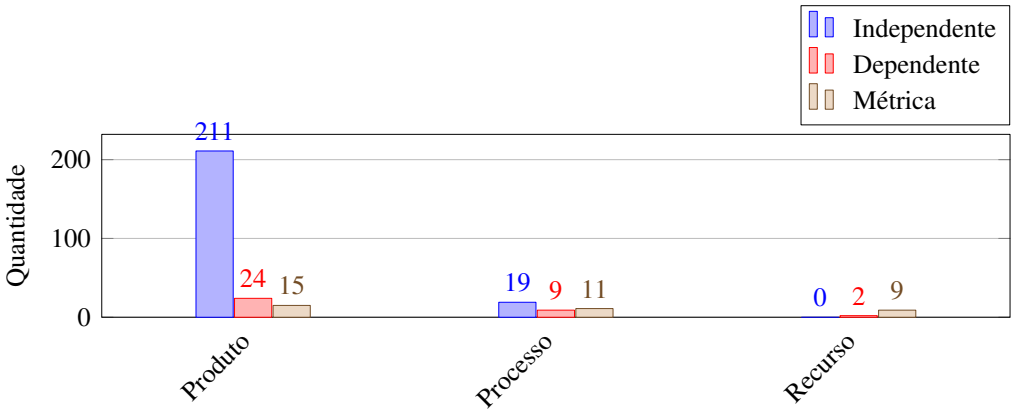
3.1.2 Medições e Perspectivas de Aplicação

Outro aspecto observado por Machado, Kemczinski e Schroeder (2023), está relacionada à categorização das medições conforme a norma ISO/IEC 15939:2007 (ISO/IEC, 2007), que as divide em variável Independente[2], Dependente[3] e Métrica[4]. A perspectiva de Produto se destaca pela quantidade de variáveis independentes na análise dos estudos, conforme demonstra Figura 7.

Além disso, cabe destacar que para a correta utilização da métrica, é necessário identificar sua perspectiva de aplicação (Produto, Processo ou Recurso). Dessa forma, tornou-se possível identificar que a perspectiva de produto é atualmente a mais relevante na manutenção de software.

[2] Independente: Medida definida por um único termo e seu método para qualificá-lo.
[3] Dependente: Medida definida pela função de dois ou mais valores das variáveis independentes.
[4] Métrica: Medida que fornece uma estimativa ou avaliação de atributos especificados.

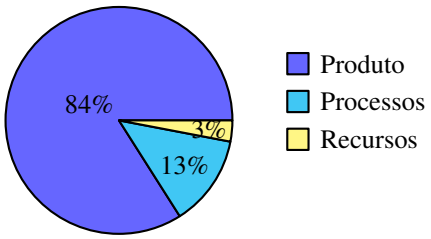
Figura 7 – Tipos de Medições



Fonte: Machado, Kemczinski e Schroeder (2023)

A Figura 8 destaca a separação dessas métricas. Dentre as 300 métricas levantadas, pode-se apontar que 251 (84%) se relacionam com Produto, 39 (13%) se relacionam com Processos e 11 (3%) se relacionam com Recursos.

Figura 8 – Quantidade de Métricas por Perspectiva



Fonte: Machado, Kemczinski e Schroeder (2023)

A perspectiva de Produto conta com a maioria das métricas da área de manutenção de *software*, podendo então tendenciar a visualização de dados através de suas métricas. Logo, uma análise e distribuição igualitária entre as perspectivas podem contribuir com avaliações mais relevantes do que apenas a avaliação isolada de apenas uma delas.

Por fim, outra abordagem relevante apontada por Machado, Kemczinski e Schroeder (2023) reside na ausência de identificação de ferramentas que adotem uma abordagem conjunta das três perspectivas de manutenção de *software* (produto, processo e recurso). Ao invés disso, as ferramentas são especializadas em apenas uma dessas perspectivas.

3.1.3 Ameaças à validade da RSL

Machado, Kemczinski e Schroeder (2023) destacam ainda ameaças a validade da pesquisa realizada, visando apontar as ações tomadas durante a pesquisa para que sua reprodutibilidade seja garantida, sendo apresentadas abaixo.

- Viés do Pesquisador: Para evitar que haja uma tendência na análise de dados por parte

dos pesquisadores, tanto o protocolo quanto a seleção dos trabalhos foram realizados pelo primeiro autor e revisados pelos demais.

- Classificação: Para evitar erros durante a classificação, a mesma foi realizada em separado pelos três pesquisadores e, após, equalizada
- *String* de busca única: Para evitar que artigos fossem incluídos indevidamente, uma vez que não foram estruturadas *strings* de busca para cada pergunta de pesquisa, na avaliação qualitativa foi adotada uma pontuação intermediária, denominada "Parcial".

3.1.4 Considerações finais da RSL

Conforme destacado por Machado, Kemczinski e Schroeder (2023) o objetivo da revisão sistemática foi o levantamento do estado atual das medições no que tange as perspectivas de manutenção de *software*. Desta forma, se pode apontar que a principal perspectiva de aplicação das métricas de manutenção de *software* é Produto, com cerca de 84% das métricas avaliadas.

No entanto, a aplicação desta pesquisa no presente trabalho se deu no levantamento das métricas propostas nas três perspectivas de aplicação, visando avaliar equitativamente a área de manutenção de *software*. Isso se deve ao fato de que Machado, Kemczinski e Schroeder (2023) apontam que a utilização apenas da perspectiva de produto, pode tendenciar análises mais aprofundadas nesta área.

Ainda, visando identificar os principais trabalhos relacionados, os 44 artigos destacados por Machado, Kemczinski e Schroeder (2023) (elencados no Apêndice B), foram analisados e separados com base nas perspectivas nos quais as métricas apontadas são destinadas, identificando os trabalhos fundados em cada uma das perspectivas.

Para isso, os trabalhos foram separados no Quadro 11, nos quais determinam os trabalhos relacionados para cada perspectiva.

Quadro 11 – Autores por perspectiva

(continua)

Autor	**Produto**	**Processo**	**Recurso**
Ahn et al. (2003)	X		
Al-Jamimi e Ahmed (2012)	X		
Bakota et al. (2011)	X		
Bhatt et al. (2006)		X	X
Börstler, Caspersen e Nordström (2015)	X	X	
Burrows et al. (2010)	X		
Çalikli, Staron e Meding (2018)	X		

Fonte: Elaborado pelo autor (2023)

Quadro 11 — Autores por perspectiva

(continuação)

Canfora et al. (2005)	X		
Choudhari e Suman (2012)			X
Concas et al. (2010)	X		
Couto et al. (2018)	X		
Dagpinar e Weber (2003)	X		
Dahab, Porras e Maag (2018)			X
Dantas, Garcia e Whittle (2012)	X		
El-Sharkawy, Krafczyk e Schmid (2020)	X		
Fioravanti, Nesi e Stortoni (1999)	X		
García, Ruiz e Piattini (2004)	X		
Gil, Goldstein e Moshkovich (2011)	X	X	
Gil, Goldstein e Moshkovich (2012)	X	X	
Gordea e Zanker (2006)		X	
Hayes, Mohamed e Gao (2003)		X	X
Hayes, Patel e Zhao (2004)	X		
Hegedüs et al. (2013)	X	X	
Hovsepyan et al. (2010)	X		
Jain, Tarwani e Chug (2016)	X		
Jørgensen e Sjøberg (2002)			X
Kulkarni, Kalshetty e Arde (2010)	X		
Lewis e Henry (1990)	X		
Li e Henry (1993)	X		
Mal e Rajnish (2014)	X		
Mamun, Berger e Hansson (2017)	X		
Ostberg e Wagner (2014)	X	X	
Razani e Keyvanpour (2019)	X	X	
Saboe (2001)	X		
Saifan e Al-Rabadi (2017)	X		
Sangwan e Singh (2013)	X		
Schnappinger et al. (2019)	X		
Selvarani, Nair e Prasad (2009)	X		

Fonte: Elaborado pelo autor (2023)

Quadro 11 — Autores por perspectiva

(conclusão)

Sharma, Bhatia e Grover (2008)	X		
Sheldon, Jerath e Chung (2002)	X		
Huang e Lai (2003)	X		
Suresh, Pati e Rath (2012)	X		
Thongmak e Muenchaisri (2009)	X		
Zhang et al. (2013)	X		
SMAE	X	X	X

Fonte: Elaborado pelo autor (2023)

Pode-se concluir que a perspectiva de Produto é predominantemente abordada na literatura disponível, com um total de 42 artigos se referindo a ela. Essa constatação sugere que a pesquisa tem direcionado sua atenção principalmente para a análise, desenvolvimento ou avaliação de características específicas de produtos.

Por outro lado, embora menos abordadas, tanto a perspectiva de Processo quanto a de Recurso também apresentam relevância, com 11 e 5 artigos, respectivamente. Isso indica que aspectos relacionados aos processos envolvidos e aos recursos empregados também têm sido objeto de interesse acadêmico, embora de forma menos acentuada.

Essa distribuição pode refletir a importância dada à análise dos produtos, ao passo que as dimensões de processo e recurso, embora menos exploradas, ainda apresentam um papel relevante no escopo das pesquisas revisadas. Portanto, essa distribuição ressalta a necessidade de um equilíbrio nas abordagens, considerando-se os diferentes aspectos para uma compreensão completa e holística do tema em estudo.

3.2 LEVANTAMENTO EXPLORATÓRIO

Diante da necessidade de identificação das métricas utilizadas em um ambiente real de manutenção de *software*, levantou-se a necessidade da aplicação de uma pesquisa, de modo a realizar a exploração de seu comportamento.

De acordo com Kitchenham, Pickard e Pfleeger (1995), as pesquisas, de modo geral, tentam capturar o que está acontecendo amplamente em grandes grupos de projetos (*research-in-the-large*), uma vez que podem ser aplicados questionários ou entrevistas a uma grande quantidade de pessoas, permitindo cobrir qualquer público-alvo.

Logo, uma pesquisa trata-se de uma investigação realizada em retrospecto, quando, por exemplo, uma ferramenta ou técnica, já está em uso há algum tempo, conforme aponta Pfleeger (1995). Ainda, Gil (2017) define a pesquisa um processo formal e sistemático de desenvolvimento do método científico, tendo seu objetivo fundamental em descobrir respostas para problemas

mediante o emprego de procedimentos científicos.

Um dos principais meio para se coletar dados qualitativos ou quantitativos, são as entrevistas ou questionários (KITCHENHAM; PICKARD; PFLEEGER, 1995), permitindo coletar uma amostra representativa da população a ser estudada. Após, os dados coletados são analisados para poderem ser derivadas conclusões descritivas e explicativas, para, por fim, serem generalizados para a população da qual a amostra foi retirada.

Seguindo o trazido por Babbie (1990), e reforçado por Wohlin, Höst e Henningsson (2003), existem três objetivos gerais para realização de uma pesquisa, sendo eles o descritivo, explicativo e exploratório.

As pesquisas descritivas têm como objetivo fazer afirmações sobre uma determinada população, identificando características ou atributos específicos, sem, no entanto, investigar as razões por trás dessas características. Por outro lado, as pesquisas explicativas buscam não apenas descrever a população, mas também explicar as razões por trás das afirmações e escolhas feitas por essa população. Por fim, as pesquisas exploratórias são conduzidas como estudos preliminares que fornecem *insights* e argumentos para pesquisas mais abrangentes e direcionadas, por meio da coleta e análise de resultados iniciais (WOHLIN; HÖST; HENNINGSSON, 2003).

Haja visto que o intuito da pesquisa a ser aplicada está em um estudo sobre o ambiente real de manutenção de *software*, com o objetivo de fornecer subsídios a uma investigação mais profunda, que é a correlação entre o ambiente real e o trazido pela literatura, foi aplicada uma pesquisa exploratória.

Desta forma, foram seguidos os 6 (seis) passos estipulados por Ciolkowski et al. (2003), sendo:

- **Definição do estudo (*Survey definition*):** Determinar o objetivo da pesquisa.
- **Desenho (*Survey Design*):** Determinar como será operacionalizado o objetivo.
- **Implementação (*Survey implementation*):** Produzir, coletar e preparar todo o material necessário para realizar a pesquisa.
- **Execução (*Survey execution*):** Coletar e processar de dados reais.
- **Análise (*Survey analysis*):** Interpretar dos dados.
- **Embalagem (*Survey packing*):** Relatar os resultados da pesquisa.

O detalhamento de cada uma das etapas será feito nas sessões seguintes.

3.2.1 Definição do Estudo

Conforme já abordado anteriormente, há uma grande pluralidade de estudos sobre métricas de *software*, bem como várias métricas são trazidos pela literatura. Contudo, não encontraram-se até o momento estudos que investiguem como estas métricas são aplicadas em ambiente real.

Desta forma, com base na RSL, foi realizado um estudo exploratório, com a finalidade de compreender as percepções da área de manutenção de *software* sobre as métricas levantados.

Esta exploração respondeu as seguintes questões de pesquisa:

- **Q1** - Qual a relação entre a demografia e perfil profissional de cada gestão com as métricas utilizadas?
- **Q2** - Qual a relação entre a tecnologia aplicada e as métricas utilizadas?
- **Q3** - Qual a relação entre as características empresariais e as métricas utilizadas?
- **Q4** - Qual a relação entre os modelos de processos e ciclos de vida e as métricas utilizadas?
- **Q5** - Qual a relação entre a maturidade do conhecimento em métricas e as métricas utilizadas?
- **Q6** - Qual a relação entre as métricas levantadas na literatura e as métricas utilizadas?

Desta forma, o objetivo foi determinar a correlação entre a literatura e a área de manutenção de *software*, permitindo determinar quais métricas seriam os mais adequadas.

3.2.2 Desenho da Pesquisa

Os autores Ciolkowski et al. (2003) determinam que o desenho da pesquisa deve demonstrar como os dados podem ser coletados e interpretados, realizando a definição do público-alvo, a modelagem conceitual dos objetos e variáveis da pesquisa, determinando como os dados serão coletados e analisados para então formular o questionário. Ainda destacam como importante a definição de questões de validade, para garantir que as respostas obtidas serão satisfatórias.

Diante disso, o público-alvo determinado para esta exploração foram os gestores de equipes de manutenção de *software*, devido ao fato de trabalharem diretamente com as métricas aplicadas a área de manutenção de *software*.

Para a operacionalização da pesquisa, foi utilizada a ferramenta Google Forms, permitindo com isso a coleta anônima das informações, bem como a extração dos dados de forma facilitada por planilhas e gráficos informativos. Logo, os dados foram coletados através de envio deste formulário diretamente para os participantes (por conveniência) que possuem algum vínculo conhecido com a área de manutenção de *software* e também publicada em mídias sociais, tais como LinkedIn e Facebook, em grupos específicos sobre o assunto de manutenção de *software*.

Os autores Wohlin, Höst e Henningsson (2003) determinam que as pesquisas exploratórias podem se utilizar de questionários pouco estruturados para realizar uma coleta e análise mais aprofundada sobre o público alvo, melhorando com isso sua a investigação. Nesta mesma linha, Ciolkowski et al. (2003) afirmam que as perguntas do questionário devem ter uma forte relação

com o objetivo da pesquisa e em pesquisas exploratórias geralmente usam perguntas abertas que podem capturar opções de resposta imprevistas.

Desta forma, para cada questão de pesquisa foi definida uma motivação, onde, a partir dela, foram levantadas as questões derivadas, utilizadas na formulação do questionário aplicado, detalhadas no Quadro 12. Esta abordagem foi utilizada, pois não foram encontrados questionários específicos na literatura para esta pesquisa exploratória.

Quadro 12 – Questões, Motivação e Questões Derivadas

(continua)

Questão	**Motivação**	**Questões Derivadas**
Q1	Determinar as relações demográficas e profissionais dos gestores	1. Qual a sua cidade/região de origem? 2. Qual a sua idade? 3. Quando foi o seu primeiro contato com a área de desenvolvimento? 4. A quanto tempo você realiza a gestão de uma área de desenvolvimento? 5. A quanto tempo você realiza a gestão de uma área de manutenção de *software*? 6. A quantos anos você considera que está programando? 7. A quantos anos você considera que NÃO está programando? 8. Atualmente qual a nomenclatura do seu cargo? 9. Qual o seu grau de instrução? 10. Costumeiramente você procura conhecimento através de quais meios?
Q2	Determinar as relações tecnológicas	11. Quais as principais linguagens de tecnologias nos quais vocês atuam? 12. Qual a principal infraestrutura utilizada?
Q3	Determinar as relações empresariais	13. Qual a sua situação de emprego atual? 14. Qual o tamanho da empresa na qual você trabalha?

Fonte: Elaborado pelo autor (2023)

Quadro 12 — Questões, Motivação e Questões Derivadas

(continuação)

Q4	Determinar os processos de desenvolvimento utilizados	15. Qual o modelo de ciclo de vida de *software* aplicado em sua empresa? 16. Qual o seu grau de familiaridade com as metodologias ágeis? 17. Qual o grau de aplicação da sua companhia com metodologias ágeis? 18. Qual a principal abordagem de desenvolvimento? 19. Se utilizam, quais das metodologias ágeis são aplicadas?
Q5	Determinar a maturidade compreendida	20. Qual o grau de maturidade de métricas de *software* você considera ter? 21. Qual o grau de maturidade de métricas de produto na sua área de manutenção de *software*? 22. Qual o grau de maturidade de métricas de processo na sua área de manutenção de *software*? 23. Qual o grau de maturidade de métricas de pessoas na sua área de manutenção de *software*?
Q6	Determinar das principais métricas quais são utilizadas pela indústria	24. Quais das métricas abaixo são aplicáveis hoje ao produto de sua área de manutenção de *software*? 25. Quais das métricas abaixo são aplicáveis hoje ao processo da sua área de manutenção de *software*? 26. Atualmente é utilizado algum dos seguintes padrões para certificação dos processos? 27. Quais das métricas abaixo são aplicáveis hoje as pessoas da sua área de manutenção de *software*?

Fonte: Elaborado pelo autor (2023)

Quadro 12 — Questões, Motivação e Questões Derivadas

(conclusão)

Q6	Determinar das principais métricas quais são utilizadas pela indústria	28. Há ainda alguma métrica não citada que não se enquadre em alguma das categorias acima que você gostaria de pontuar?

Fonte: Elaborado pelo autor (2023)

As respostas para as questões derivadas referentes a questão principal foram extraídas da RSL abordada na Seção 3.1, de maneira a permitir a correlação entre a literatura e o ambiente real de manutenção de *software*.

Por fim, de maneira a avaliar a validade das respostas, foram elaboradas as seguintes perguntas: "O que você acha da duração da pesquisa?"e "Quão fácil ou difícil foi preencher esta pesquisa?", bem como foi considerado o tempo de resposta de cada gestor, para determinar o engajamento das respostas.

3.2.3 Implementação e Execução da Pesquisa Exploratória

No período de 18 de outubro a 05 de novembro de 2021, foi realizada a coleta de dados por meio do questionário, referenciado no Apêndice C. Inicialmente, foram convidadas pessoas ligadas a área de manutenção de *software*, independente de cargo ou posição hierárquica, de maneira individual e direta.

Em sequência, foram convidados através de publicações em mídias sociais, gerentes e coordenadores da área de manutenção de *software* para que os mesmos pudessem responder ao questionário. As publicações foram realizadas em grupos pertinentes a área de manutenção de *software*.

Desta forma, o questionário contou com um total de 37 respostas, onde todos os participantes aceitaram o termo de consentimento livre esclarecido (Apêndice A) para publicação das informações.

3.2.4 Análises e Resultados

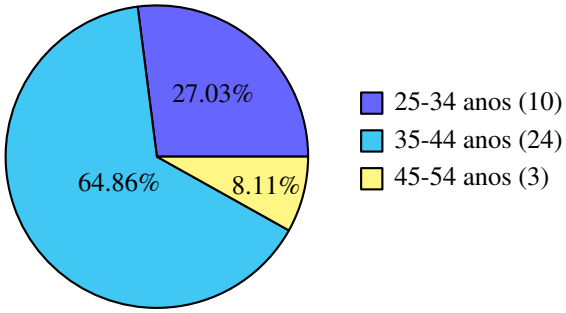
Após o encerramento da coleta de dados, os mesmos foram sumarizados e categorizados, de maneira a compreender como ocorreu a distribuição destas informações. Ainda, de maneira a responder às questões de pesquisa, os resultados foram analisados sob esta perspectiva.

3.2.4.1 Q1 - Qual a relação entre a demografia e perfil profissional de cada gestão com as métricas utilizadas?

Conforme destacado pela Figura 9, a idade média dos participantes se concentrou em torno de 35 a 44 anos, o que destaca um público relativamente jovem. Ainda, a Figura 10 destaca

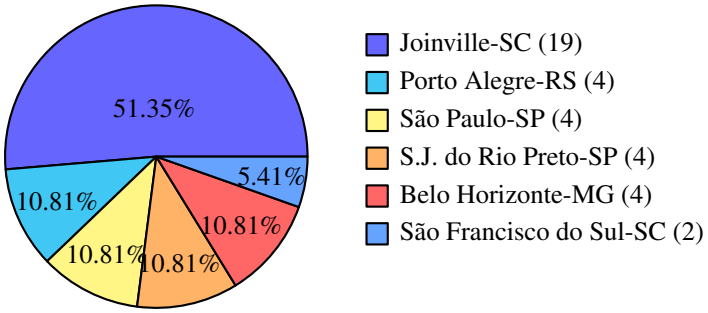
que os respondentes se concentram na cidade de Joinville.

Figura 9 – Idade Média dos Respondentes



Fonte: Elaborado pelo autor (2023).

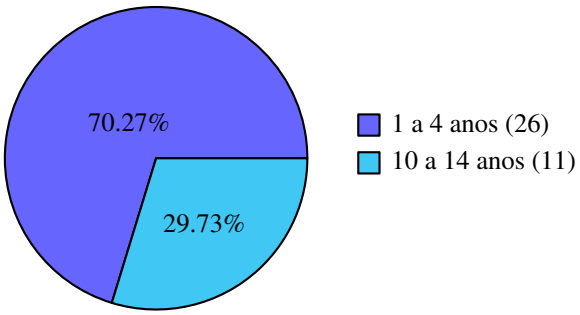
Figura 10 – Localidade dos Respondentes



Fonte: Elaborado pelo autor (2023).

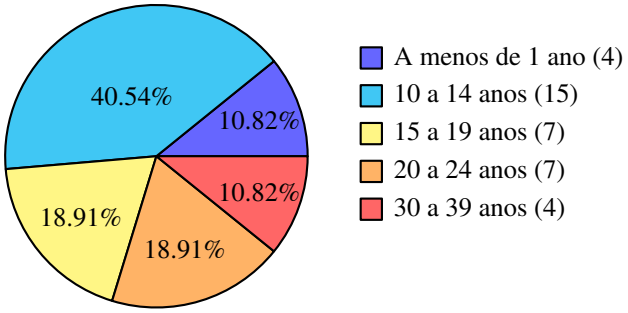
Outro fator interessante avaliado está no tempo no qual atuam de forma direta ou indireta na área de gestão de manutenção de *software*, conforme destaca a Figura 11, levando a concluir que são gestores novos na área e que possuem uma certa vivência na área de manutenção, conforme destaca a Figura 12.

Figura 11 – Tempo de Gestão na Área de Manutenção



Fonte: Elaborado pelo autor (2023).

Figura 12 – Primeiro Contato com a Área de Manutenção



Fonte: Elaborado pelo autor (2023).

Por fim, uma característica que auxilia também no conhecimento da área é que cerca de 70% possuem formação mínima em Pós-Graduação (*lato sensu* ou *stricto sensu*) e que a grande maioria tende a procurar atualização profissional em cursos e certificações online.

Desta forma, caracteriza-se o grupo respondente por gestores novos, tanto em idade quanto em experiência na área de gestão, que já atuam por um período médio de 10 a 14 anos na área de manutenção de *software* e possuem um grau de escolaridade elevado. Logo, possuem uma bagagem de métricas práticas e teórica grande, no qual pode auxiliar em sua aplicação.

3.2.4.2 Q2 - Qual a relação entre a tecnologia aplicada e as métricas utilizadas?

Foi identificada que aproximadamente 75% dos respondentes possuem uma relação com as tecnologias em nuvem e com a aplicação de sistemas em nuvem, determinando que as novas tecnologias vêm ganhando força na indústria.

As linguagens e *frameworks* que merecem destaque são JavaScript, com 16,36%, Java, com 12,72%, Node, com 12,72%, e C#, com 9,09%.

As tecnologias e estruturas aplicadas, por não estarem diretamente ligadas as métricas trazidas pela literatura, seja pelo tempo no qual esta foi proposta ou pela incompatibilidade tecnológica, fazem com que as métricas não sejam efetivamente aplicadas. Desta forma, identificar quais métricas podem ser aplicadas à área de manutenção de *software* pode auxiliar os gestores na tomada de decisão desta área.

3.2.4.3 Q3 - Qual a relação entre as características empresarias e as métricas utilizadas?

Com relação a características empresariais, não houve uma grande relevância no que tange ao tamanho de empresa, pois a distribuição de respostas foi equitativa. Já no que tange ao regime de contratação, quase 97% são empregados em tempo integral.

Logo, não é possível determinar se há uma relação clara entre o tamanho da empresa e as métricas aplicadas, nem se o regime de contratação pode ter alguma influência.

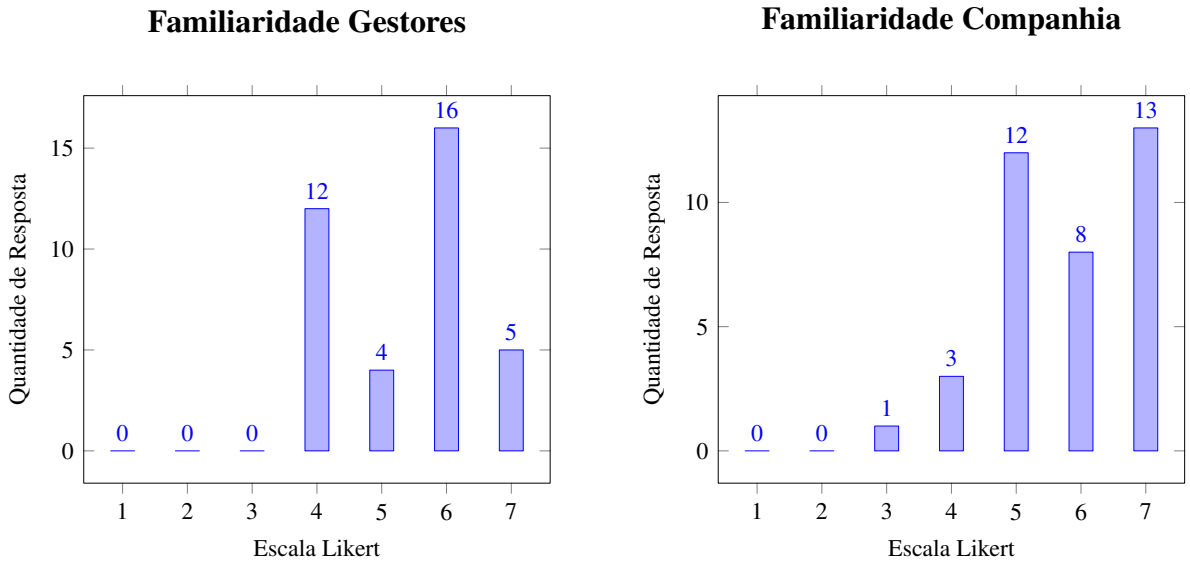
3.2.4.4 *Q4 - Qual a relação entre os modelos de processos e ciclos de vida e as métricas utilizadas?*

Para avaliar o grau de maturidade do conhecimento em métricas dos respondentes, foi aplicada uma escala Likert de 7 pontos, variando de 1 como pouco familiar e 7 naturalmente aplicável.

Uma resposta em escala de Likert (MURRAY, 2013) é uma ferramenta utilizada com mais frequência para investigar como um determinado item é avaliado, através de uma escala numerada onde é possível realizar uma valoração, como, no caso acima, 1-2-3-4-5-6-7.

Se destaca que 62,16% dos respondentes utilizam o ciclo de vida evolutivo e, tanto eles quanto suas empresas, possuem uma familiaridade alta com as metodologias ágeis, conforme Figura 13, o que confirma sua alta aplicabilidade na indústria atualmente.

Figura 13 – Familiaridade com Metodologia Ágil



Fonte: Elaborado pelo autor (2023).

Entretanto, a percepção dos gestores é que suas companhias estão mais familiarizadas com a aplicação das metodologias ágeis do que eles próprios. Esta discrepância fica evidente ao analisar as médias das respostas, sendo a média dos gestores de 5,38, com um desvio padrão de 1,09, enquanto a média de suas empresas é de 5,78, com um desvio padrão de 1,11. Observa-se que a variância nas respostas é relativamente baixa, mas a tendência é que seja maior na avaliação da empresa.

Isto pode ocasionar uma diferença na utilização de métricas, haja visto que as empresas podem aplicar métricas pouco conhecidas ou pouco utilizadas pelos gestores, denotando a importância de uma normalização nesta área.

Por fim, a implementação da metodologia ágil resulta em um afastamento das métricas convencionais, seja devido ao período em que foi concebida, conforme previamente destacado, ou

devido às discrepâncias nas atividades e processos propostos. Portanto, torna-se necessária uma reavaliação das métricas aplicadas em contraste com o conhecimento disponível na literatura.

3.2.4.5 Q5 - Qual a relação entre a maturidade do conhecimento em métricas e as métricas utilizadas?

Para as métricas de forma geral, os respondentes consideraram que sua maturidade é médio-superior, já para métricas de produto e processo consideraram que sua maturidade é média.

Entretanto, para as métricas de recursos, a maturidade considerada foi médio-inferior, indicando que podem ainda ser pouco exploradas pela indústria, bem como esta informação corrobora com a revisão sistemática realizada, pois as métricas de recursos foram as menos identificadas.

3.2.4.6 Q6 - Qual a relação entre as métricas levantadas na literatura e as métricas utilizadas?

Para avaliar a relação entre as métricas levantadas pela revisão sistemática e as métricas utilizadas pela indústria, foram questionadas quais das 10 (dez) métricas destacadas em produto, processo e recurso são aplicadas.

Sobre as métricas de produto, 8 (oito) são aplicadas de alguma forma na indústria, podendo-se destacar LoC — Linhas de Código (9,10%), NOA — Número de Atributos (16,78%), RFC — Respostas para a Classe (16,78%) e N — Tamanho do Programa (9,10%).

Já sobre as métricas de processo, 7 (sete) são aplicadas de alguma forma na indústria, podendo-se destacar ASP - Pontos de História Ajustados (17,09%), DOC - Atualização de Documentação (17,09%), TST - Testabilidade (17,09%) e MI - Índice de Manutenção (13,68%).

Por fim, das métricas de recurso, 7 são aplicadas de alguma forma na indústria, com destaque para E - Esforço (40%) e D - Dificuldade (25%).

Ainda, foi questionado de forma aberta se existiria alguma métrica aplicada que não estava elencada nas anteriores, contudo esta questão não teve nenhuma resposta.

Com isso, afirma-se que as métricas levantadas são utilizadas pela indústria, entretanto não há um padrão pré-estabelecido para sua aplicação, haja visto que nenhuma das respostas foram assinaladas de forma massiva.

Com isso que não há diferenças significativas entre as métricas levantadas pela literatura e aplicadas pela indústria, contudo a falta de padronização prejudica uma análise mais aprofundada.

3.2.4.7 Feedback

De maneira a determinar o engajamento dos respondentes com suas respectivas respostas, foram verificados os tempos de respostas e realizados questionamentos a cerca do nível de perguntas e duração.

O tempo médio de respostas foi de 22 minutos, o que indica que os respondentes possivelmente estavam focados nas respostas e se aplicaram em suas asserções, fornecendo respostas com qualidade.

Ainda, 81,08% dos respondentes acharam a duração apropriada e 70,27% acharam o nível de questionamento mediano, ou seja, não acharam nem difícil e nem fácil sua resposta, o que determina que os mesmos não foram prejudicados por uma eventual extensividade do questionário.

3.2.5 Ameaças à validade da Pesquisa Exploratória

Como já destacado anteriormente, para garantir a qualidade da pesquisa realizada, é necessário o levantamento das ameças a validade da pesquisa durante sua realização, visando controlar as ameaças e tomando ações necessárias para contorná-las, de maneira a orientar a reprodutibilidade da pesquisa.

3.2.5.1 Viés do Pesquisador

A pesquisa foi planejada e realizada pelos mesmos pesquisadores, desta forma, tanto as questões de pesquisa, as questões do questionário, quanto as análises realizadas podem ter sofrido um direcionamento, contudo, para mitigar isso:

- Foram feitas duas análises separadas das repostas, uma simplificada, de forma a verificar o quantitativo de respostas, e outra realizando o relacionamento entre as respostas e as questões de pesquisa, para serem identificadas possíveis distorções durante a análise dos resultados;
- As análises das respostas foi realizada pelo autor, revisada pelos seus orientadores.

3.2.5.2 Tendência nas Respostas

Os respondentes podem tendenciar suas respostas ao que gostariam de aplicar em detrimento ao que aplicam justamente por estarem sendo avaliados. Devido a não aparentarem possuir respostas enviesadas e, também, pela qualidade das respostas nos questionários, os pesquisadores optaram por aceitar as respostas como válidas.

3.3 CONSIDERAÇÕES DO CAPÍTULO

O processo de revisão sistemática da literatura resultou em 2145, dos quais 44 atenderam aos critérios de inclusão e exclusão e abordavam as métricas referentes a área de manutenção de *software*. Foram identificadas as 10 (dez) métricas de produto, processo e recursos, que são mais destacadas pela literatura, o que foi possível determinar as métricas mais relevantes.

Ainda, por meio da pesquisa exploratória, aplicada com base em um questionário, foi possível determinar a correlação das métricas identificadas e sua real aplicação na indústria, podendo então fazer uma comparação com o que a literatura nos trouxe e o que é aplicado dentro do contexto real.

Esta análise nos permitiu identificar que as métricas trazidas pela literatura, são aplicadas pela indústria, porém não é possível determinar uma padronização destas métricas, pois não foi identificado um padrão de aplicação entre os respondentes da análise exploratória. O que pode caracterizar essa não padronização é o descompasso entre a evolução da área de manutenção de *software* e as métricas aplicadas, tendenciando aplicações defasadas ou ainda inefetivas, conforme traz Frantz et al. (2019).

Também, foi possível identificar que um *framework* possibilitaria ao gestor da área de manutenção de *software* avaliar todos os aspectos e perspectivas da área, auxiliando na tomada de decisão e trazendo maior padronização ao seu processo e maturidade na aplicação de métricas.

Nenhum dos artigos analisados na RSL trouxe diretamente um rol de métricas que possa ser aplicada a área de manutenção de *software*, bem como não fornecem nenhuma ferramenta apropriada para realizar esta avaliação. As ferramentas trazidas durante o levantamento, por mais que tragam avaliações e validações de métricas, nenhuma delas compõem às três perspectivas da manutenção, o que corrobora com a necessidade de uma ferramenta que realize a análise conjunta das perspectivas de produto, processo e recurso, pois, conforme destacado por Molnar, Neamtu e Motogna (2020), estas perspectivas influenciam diretamente a entrega do *software*.

Logo, analisando a questão de pesquisa: "Como melhorar as entregas das atividades de manutenção de *software*, reduzindo seu tempo, custo e otimizando seu trabalho?", através da RSL, se percebe que há uma necessidade latente na área de manutenção de *software* no que tange a avaliação de todas as suas perspectivas, pois, hoje, nenhum *software* ou ferramenta realiza este trabalho. A proposta, descrita no próximo capítulo, envolve a implementação de um *framework* que visa justamente avaliar às três perspectivas em conjunto, de maneira a auxiliar na tomada de decisão na área de manutenção de *software*.

4 SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS ATIVIDADES NA ÁREA DE MANUTENÇÃO DE SOFTWARE

O SMAE, acrônimo para *Software Maintenance Activity Evaluation*, é um *framework* com o objetivo de fornecer suporte aos gestores da área de manutenção de *software*. Sua finalidade é a avaliação das métricas com base nas perspectivas de produto, processo e recurso, e com foco na priorização e alocação de tarefas, que tem em vista facilitar a tomada de decisão na priorização e agilidade na realização das tarefas.

As etapas específicas da DSRM são: Identificação do Problema, Definição dos Objetivos, Projeto e Desenvolvimento, Demonstração, Avaliação e Comunicação. A fase de "Identificação do Problema" foi contextualizada e definida nos Capítulos 2 e 3, visando identificar as métricas[1] aplicadas à área de manutenção de *software*. Com base nos trabalhos relacionados e na fundamentação teórica adotada para o desenvolvimento de um sistema de apoio aos gestores da área de manutenção de *software* que auxilie as priorizações e alocações de tarefas, foram identificadas as seguintes lacunas:

- Ausência de grupos de métricas que se apliquem de forma padronizada à área de manutenção de *software*, ou seja, não foi possível identificar um conjunto de métricas que hoje são comumente aplicadas a área de manutenção de *software*. Isso é percebido durante a Pesquisa Exploratória abordada no Capítulo 3, pois não foram identificadas métricas utilizadas por todos, ou pela maioria (mais de 50%), dos gestores da área de manutenção de *software* que responderam à pesquisa. Na Revisão Sistemática da Literatura, também não foi identificada pluralidade de métricas aplicadas à área de manutenção de *software*;

- Embora tenham sido desenvolvidos estudos voltados ao levantamento de métricas de *software*, tais como a de Meidan et al. (2018), Chug e Malhotra (2016) e Wu et al. (2016), nenhum deles se dedica exclusivamente às métricas de manutenção de *software* sob as perspectivas de produto, processo e recurso;

- Não há, ainda, uma ferramenta que permita a visualização das métricas de manutenção de *software* sob a ótica de produto, processo e recurso, conforme abordado no Capítulo 3. Durante a Revisão Sistemática da Literatura foram identificadas as ferramentas utilizadas para controle das variáveis e métricas da área de manutenção de *software*, contudo, não foi possível identificar nenhuma ferramenta que permita o controle das três perspectivas, apenas ferramentas especializadas em uma delas;

Na sequência, a etapa de Definição dos Objetivos, bem como a etapa de Projeto e Desenvolvimento são abordadas nas Seções 4.1 e 4.2. Já as etapas de Demonstração e Avaliação

[1] Medida que fornece uma estimativa ou avaliação de atributos especificados. Serve de base para a tomada de decisões, de maneira a quantificar ou precisar valores que por si só podem ser imperfeitos, ou seja, que sozinhos, muitas vezes, não trazem todas as informações necessárias.

são discutidas no Capítulo 5, cujo objetivo é detalhar a experimentação da ferramenta e os grupos de usuários nos quais ela foi aplicada.

Por fim, a Comunicação se dá por meio da publicação desta pesquisa, fornecendo visibilidade à problematização avaliada e a sua importância, ao artefato, sua utilidade e novidade, ao rigor de seu design e seu nível de eficácia para pesquisadores e outros públicos relevantes, como profissionais atuantes na área de manutenção de *software* (PEFFERS et al., 2007).

4.1 OBJETIVO DO *FRAMEWORK*

Conforme descrito no Capítulo 3, por mais que os trabalhos relacionados possuam foco em métricas de *software*, poucos são direcionados à área de manutenção de *software* ou sequer possuem uma análise conjunta das perspectivas de produto, processo e recurso dessa área. Logo, se faz necessário uma análise mais aprofundada no que tange a uma avaliação conjunta dessas métricas da área de manutenção de *software*, de maneira a fornecer suporte aos gestores, auxiliando na priorização e alocação de recursos nas tarefas correspondentes a essa área.

Desse modo, a proposta inicial, necessária para a etapa de Definição dos Objetivos trazida por Peffers et al. (2007), está na criação do referido *framework*, que aborda as lacunas previamente identificadas, oferecendo um meio de análise integrada das perspectivas relacionadas ao produto, ao processo e ao recurso, mediante a utilização de um conjunto de métricas preestabelecido.

É importante reforçar que os trabalhos relacionados possuem fundamental relevância no que tange à fundamentação das métricas propostas, bem como o papel da indústria como validador das métricas para a produção de um *framework* consistente.

Cabe destacar que um dos diferenciais para a análise da área de manutenção de *software* é uma avaliação de forma conjunta das perspectivas de produto, de processo e de recurso, denominada de avaliação global, considerando de maneira igualitária estas perspectivas. Dessa forma, não há foco para nenhuma perspectiva específica, o que possibilita ter mais assertividade sobre a área de manutenção de *software*, haja visto que, conforme trazido por April e Abran (2008) e Iqbal et al. (2021), a avaliação da área de manutenção de *software* deve ocorrer considerando as três perspectivas.

Outro ponto de melhoria conceitual está na consideração dos diferentes tipos de atividades da área de manutenção de *software*, conforme elencados na Seção 2.4.3 e mencionados por Wazlawick (2013) e Jones (1998). Essas atividades possuem diferenças em seus processos de execução, logo, o *framework* permite que cada processo distinto da área de manutenção de *software* seja detalhado, de maneira que os gestores possam acompanhar a evolução de cada atividade.

Por fim, o *framework* sugere uma possível composição de equipe para atuação em uma determinada atividade, que é realizado com base nos tipos de atividade e recursos disponíveis, cruzando essas informações de maneira a identificar os melhores recursos, considerando as métricas estimadas a serem alocadas para cada etapa da atividade, cujo objetivo é auxiliar o

gestor no planejamento e na distribuição das atividades.

Adicionalmente, o *framework* proposto foi instanciado em um artefato computacional por meio de um sistema que possui uma identidade visual própria, possibilitando a gestão de informações, bem como oferecendo uma *Application Programming Interface* - Interface de Programação de Aplicativos (API) que permite a integração com outros sistemas.

Por fim, cabe destacar que no levantamento exploratório, tanto bibliográfico quanto no questionário com os gestores, as métricas tiveram foco no paradigma de orientação a objeto, dessa forma, o *framework* está direcionado a atuações nesta abordagem de desenvolvimento.

4.2 ESTRUTURA DO *FRAMEWORK*

A próxima etapa do DSRM a ser seguida é o Projeto e Desenvolvimento, que consiste na elaboração de um desenho de solução a partir do problema identificado (PEFFERS et al., 2007). Dessa forma, a estrutura do *framework* foi concebida de maneira a compor as perspectivas de produto, de processo e de recurso, bem como as atividades de Reparação de Defeitos, Remoção de Módulos Falhos, Suporte a Usuários, Migração entre Plataformas, Conversão de Arquitetura, Adaptações Obrigatórias, Otimização de Performance e Melhorias que compõem a área de manutenção de *software*.

Para isso, foram considerados os cálculos das métricas identificadas nos trabalhos da Revisão Sistemática da Literatura apresentada no Capítulo 3, compostos por valores separados para cada perspectiva. Esses valores são determinados conforme a base de cálculo da métrica, desse modo, métricas ascendentes, ou seja, métricas nas quais os maiores valores são melhores para a manutenção de *software*, terão valores mais elevados quando, no cálculo, atingirem graus mais elevados, bem como métricas descendentes, ou seja, métricas cujo os menores valores são melhores para a manutenção de *software*, terão valores mais elevados quando, no cálculo, atingirem graus mais baixos. Um exemplo desse processo é descrito na Seção 4.2.1.

Ao final, é feita uma média aritmética de cada métrica, pois, conforme destaca Lilja (2005), trata-se da melhor forma de obter um valor proporcional ao total calculado, permitindo, com isso, determinar o que denominamos “Grau da Perspectiva”. Também é graduado o que denominamos “Grau da Tarefa”, que é a soma percentual dos valores de cada perspectiva.

Inicialmente, as perspectivas são distribuídas equitativamente, como já apontado anteriormente, em 34% para Produto, 33% para Processo e 33% para Recursos. Com isso não há interferência de uma perspectiva sobre a outra, sendo que o valor final resultante é proporcional a cada perspectiva, visando à solução de uma das lacunas identificadas, em que é possível tanto uma análise separada das perspectivas de produto, de processo e de recurso, quanto uma visão unificada dessas perspectivas.

4.2.1 Exemplo de Cálculo

De maneira a exemplificar a estrutura de cálculo a ser realizada, foram considerados cenários hipotéticos de situações da área de manutenção de *software* envolvendo produto, processos e recursos.

Dessa forma, como produto da manutenção de *software*, foi considerado o cenário descrito no Quadro 13:

Quadro 13 – Exemplo de Métricas de Produto

Programa	Função	LOC	WMC	DIT	COB	RFC	NOC	LCOM	N	CC	NOA
A	Z	400	3	1	2	4	2	3	4	15	6
A	Y	100	3	1	2	4	0	3	4	3	2
A	X	250	3	1	2	4	0	3	4	3	2
B	W	300	2	1	1	1	0	1	4	6	1
C	V	50	1	1	1	1	0	1	4	7	1

Fonte: Elaborado pelo autor (2023).

O *framework* utiliza as 10 métricas relativas a produto, identificadas na Revisão Sistemática da Literatura, de maneira a comporem as métricas de produto do *framework*. Essas métricas são: LOC - Linhas de Código, WMC - Método Ponderado por Classe, DIT - Profundidade na Árvore de Herança, CBO - Acoplamento entre classes, RFC - Respostas para a Classe, NOC - Número de Filhos, LCOM - Falta de Coesão de Métodos, N - Tamanho do Programa, CC - Complexidade da Classe e NOA - Número de Atributos.

Também, como processo da manutenção de *software*, foi considerado o cenário descrito no Quadro 14:

Quadro 14 – Exemplo de Métricas de Processo

Processo	Etapa	MI	NPR	NDA	NDWP	NDWPIn	NDWPOut	ASP	DOC	TST	SES
P1	E1	10	1	3	2	3	2	2	0	1	2
P!	E2	15	1	3	2	1	0	2	0	1	2
P1	E3	20	1	3	2	2	3	3	2	1	2
P2	E4	-5	1	1	1	3	1	2	1	5	2
P2	E5	0	1	1	1	2	1	4	1	1	2

Fonte: Elaborado pelo autor (2023).

O *framework* utiliza as 10 métricas relativas a processo, identificadas na Revisão Sistemática da Literatura, de maneira a comporem as métricas de processo do *framework*. Essas métricas são: MI - Índice de Manutenibilidade, NPR - Número de funções que participam do processo, NDA - Número de dependências de precedência entre atividades, NDWP - Número de dependências entre produtos de trabalho e atividades, NDWPIn - Número de dependências de entrada dos produtos de trabalho com as atividades no processo, NDWPOut - Número de dependências de saída dos produtos de trabalho com as atividades no processo, ASP - Pontos de História Ajustada, DOC - Atualização de Documentação, TST - Testabilidade e SES - Conformidade com os padrões de engenharia de *software*.

Por último, como recurso da manutenção de *software*, foi considerado o cenário descrito no Quadro 15:

Quadro 15 – Exemplo de Métricas de Recurso

Recurso	E	D	BDD	CONF	DEVMSYS	DEVSYS	ESS	FPL	MEXPAPP	MEXPTOT
R1	8	3	3	1	1	2	2	2	1	2
R2	4	2	2	2	1	1	1	2	2	1
R3	6	1	1	3	3	3	3	2	3	3

Fonte: Elaborado pelo autor (2023).

O *framework* utiliza as 10 métricas relativas a recurso, identificadas na Revisão Sistemática da Literatura, de maneira a comporem as métricas de recursos do *framework*. Essas métricas são: E - Esforço de Trabalho, D - Dificuldade de Desenvolvimento, BDD - Complexidade de domínio do sistema, CONF - Confiança na resolução de tarefas, DEVMSYS - Experiência de desenvolvimento de módulo, DEVSYS - Experiência total de desenvolvimento, ESS - Experiência com *software*, FPL - Familiaridade com a linguagem de desenvolvimento, MEXPAPP - Experiência de manutenção na aplicação, MEXPTOT - Experiência total de manutenção.

O cálculo do *framework* inicia-se com a criação de uma nova tarefa para a área de manutenção de *software*, na qual são definidas as informações necessárias para sua realização, como a atividade a ser executada, os produtos que sofrerão alterações e os indicadores de processo, como o ASP.

Com base nessas informações, o *framework* avalia as graduações de cada perspectiva com o propósito de determinar o grau de importância dessa tarefa para a equipe de manutenção de *software*.

É importante salientar que, considerando os cenários previamente mencionados, cada métrica apresenta uma base de cálculo distinta. Portanto, a primeira etapa a ser realizada é a conversão de todos os valores para uma base de cálculo comum. Essa padronização é alcançada por meio da transformação de todos os valores em percentagens, possibilitando a combinação matemática das métricas.

Para essa transformação, é aplicado a variação percentual (VP) em cada métrica, identificando, com isso, um peso percentual para cada métrica utilizada. O cálculo da VP é representado pela seguinte expressão:

$$VP = (((M - Min)/(Min - Max)) * 100) * (-1)$$

Os valores são definidos como:

- M — Valor da métrica a ser ponderada;
- Max — Valor máximo da respectiva métrica no conjunto de dados;
- Min — Valor mínimo da respectiva métrica no conjunto de dados;

Contudo, o cálculo da VP sempre fornece valores escalonados do menor percentual para o maior percentual. Por exemplo, em uma escala de 0 a 10, os valores de VP são, respectivamente, 0 e 100%. Diante disso, ainda nessa etapa é importante identificar o contexto de aplicação da métrica. Esse contexto considera duas afirmações:

- **A1** — Quanto maior a métrica, maior a sua importância;
- **A2** — Quanto menor a métrica, maior a sua importância;

Dessa forma, métricas como LOC, N e CC são melhores quando possuem valores menores e se enquadram na afirmação **A2**. Já as métricas como DOC, CONF e FPL são melhores quando possuem valores maiores e se encaixam na afirmação **A1**.

Com a explanação do contexto supracitado, é possível inverter as grandezas nas métricas da afirmação **A2**, de modo a obter os menores valores com os maiores percentuais. Essa inversão é realizada calculando-se o percentual contrário ao obtido. Portanto, se um percentual de 25% é calculado para VP nas métricas **A2**, a inversão desse valor seria 100% - 25%, resultando 75%. Essa abordagem contribui para a interpretação adequada das métricas e seu impacto na avaliação das tarefas de manutenção de *software*.

Para exemplificar os cálculos, são demonstradas, no Quadro 16, algumas métricas com a definição de seu contexto e aplicação da VP. Para valores são utilizados os exemplos dos Quadros 13, 14 e 15.

Quadro 16 – Aplicação da VP e Inversão de Grandezas

Perspectiva	**Métrica**	**Contexto**	Min	Max	N	VP
Produto	LOC	A2	50	400	250	42,86%
Produto	CC	A2	3	15	6	75%
Processo	MI	A1	-5	20	10	60%
Processo	DOC	A1	0	2	1	50%
Recurso	CONF	A1	1	3	3	100%
Recurso	ESS	A1	1	3	0	0%

Fonte: Elaborado pelo autor (2023).

Após a obtenção de todas as VP das métricas, estas são somadas, sendo então aplicada uma média aritmética. Cada perspectiva possui seu somatório de métricas e média, com notação específica[2]. A representação desse cálculo é dada pela seguinte expressão:

[2] Para a perspectiva de produto, será utilizada a notação $Prod_p$, para a perspectiva de Processo, será utiliza a notação $Porc_p$ e para a perspectiva de Recurso, será utilizada a notação Rec_p

$$\frac{1}{n}\sum_{i=1}^{n} VP_i$$

Esse cálculo determina o "Grau da Perspectiva", indicando qual a relevância daquela informação nas combinações disponíveis no conjunto de dados. Cabe destacar que os cálculos referentes a cada perspectiva podem variar conforme a atividade a ser executada, os produtos associados à tarefa e os recursos disponíveis para realização.

Com a obtenção das médias aritméticas, é possível calcular o "Grau da Tarefa" (*GT*). Através do grau de cada perspectiva é feita uma distribuição nos percentuais previamente estabelecidos, sendo 34% para Produto[3], 33% para Processo[4] e 33% para Recurso[5]. Portanto, é possível determinar o grau de dificuldade para a realização de uma tarefa com base em todas as perspectivas da manutenção de *software*. A expressão que representa o cálculo é a seguinte:

$$GT = (Prod_p * Prod_g) + (Proc_p * Proc_g) + (Rec_p * Rec_g)$$

Adicionalmente, para determinar a sugestão de alocação de recursos, é efetuado o cálculo da matriz de combinações de aplicação desses recursos, com base na atividade da tarefa. A partir da atividade, são identificadas as etapas necessárias para a realização da tarefa, bem como os recursos que podem ser alocados para cada etapa.

Em seguida, é calculado o $Proc_p$ para cada possível combinação de recursos. Como sugestão para a alocação na tarefa, é selecionada a combinação com o maior valor de $Proc_p$. Essa abordagem determina, com base nas métricas, quais os melhores recursos disponíveis para a execução da tarefa.

Por fim, para o cálculo do grau de dificuldade da tarefa (*GT*), o valor que representa $Proc_p$ corresponde ao valor da sugestão de alocação, identificando, com isso, a graduação da tarefa com base na melhor alocação dos recursos disponíveis.

4.3 ESTRUTURA DA FERRAMENTA

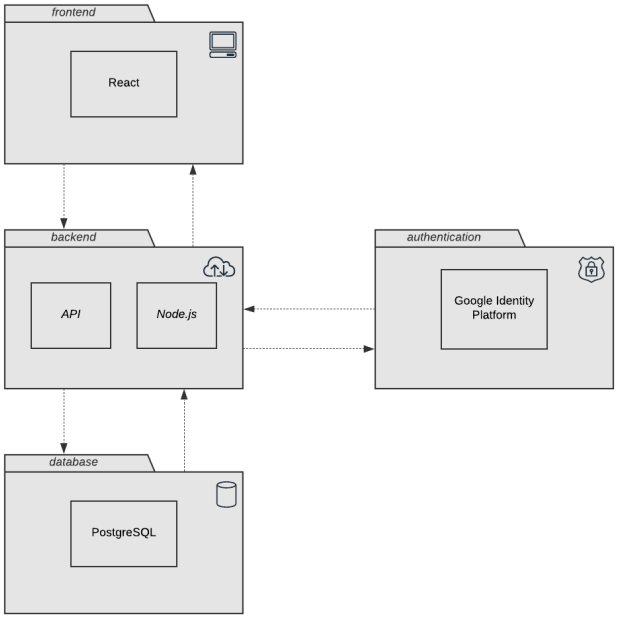
Para efetuar a avaliação do *framework*, e dar continuidade às etapas do DSRM proposto por Peffers et al. (2007), foi criada uma aplicação com a capacidade de gerenciar as informações por meio de uma interface visual. Além disso, a ferramenta possibilita o acesso integral à estrutura subjacente do *framework* através de um conjunto de APIs.

[3] A notação dessa graduação é $Prod_g$
[4] A notação dessa graduação é $Proc_g$
[5] A notação dessa graduação é Rec_g

Conforme já destacado na Seção 3.1.2 do Capítulo 3, foram identificadas ferramentas utilizadas para o controle das métricas da área manutenção de *software*, contudo, nenhuma que se destaque em sua utilização.

Outro ponto relevante está no fato de que as ferramentas são programas instaláveis ou de extensões para instalação, logo, a utilização de uma ferramenta via API é uma inovação para a área. Além disso, a arquitetura foi idealizada para poder ser utilizada em servidores de *Cloud*, reduzindo. assim, seu custo. Para melhor esclarecer, a arquitetura proposta é exibida na Figura 14.

Figura 14 – Arquitetura do *framework*



Fonte: Elaborado pelo autor (2023).

A arquitetura foi dividida em quatro componentes que possuem atribuições distintas. O componente de *backend* se integra com o componente de *database* e contém toda a regra de negócio aplicada no *framework*, e o componente *frontend* disponibiliza a interface de usuário. Por fim, o componente *authentication* realiza a validação de autenticação. As funcionalidades de cada componente são descritas a seguir:

- *backend*: responsável pela centralização da regra de negócio, externalizando a API de comunicação com o *frontend* e demais ferramentas. Integra com a *database* e com a *authentication*. Sua construção foi feita com Node.js[6];
- *database*: o sistema de gerenciamento de banco de dados (SGBD) utilizado pelo *framework* é relacional, utilizando a tecnologia PostgreSQL que é desenvolvido como

[6] Conforme seu site oficial (https://nodejs.org/pt-br/about/), o Node.js é um ambiente de execução JavaScript assíncrono orientado a eventos e projetado para o desenvolvimento de aplicações escaláveis de rede.

projeto de código aberto. A escolha desse banco se deu por sua agilidade e por possuir o conceito ACID, que é um conjunto de propriedades de transações de banco de dados, que visa garantir a confiabilidade nas transações;

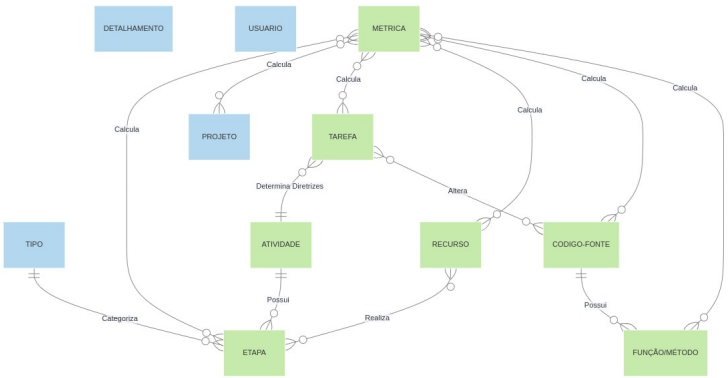
- *frontend*: a interface desenvolvida é *web*, pois tem o objetivo de permitir o acesso tanto em navegadores *desktop* quanto *mobile*. Foi utilizada a tecnologia *React.js*, pois em uma pesquisa feita por Stack Overflow (2022), realizada com mais de 80 mil desenvolvedores, ela representa o *framework web* mais comumente utilizado, bem como o autor já detém conhecimento nesse *framework*;

- *authentication*: para garantir a integridade, foram utilizados as autenticações via ferramenta social, sendo que a ferramenta escolhida foi o *Google Identity Platform*. A utilização de ferramenta social auxilia na simplificação de acesso para o usuário, que permite a utilização de várias plataformas através de um único acesso. Também foi desenvolvida uma autenticação simples da própria ferramenta para garantir acesso àqueles que não possuem autenticação com a plataforma *Google*.

4.4 DOMÍNIOS

De maneira a identificar os principais aspectos do domínio relacionado ao *framework*, foi elaborado um diagrama entidade relacionamento, conforme apresentado na Figura 15, que tem o objetivo de registrar as informações necessárias para os cálculos das métricas e para as sugestões de composição de equipe. As entidades destacadas em verde correspondem às informações básicas das perspectivas, já as destacadas em azul correspondem às informações complementares que auxiliam na construção do cálculo.

Figura 15 – Diagrama Entidade Relacionamento



Fonte: Elaborado pelo autor (2023).

As entidades do sistema são relacionadas com Recurso Humano, Atividade, Etapa, Tarefa, Código Fonte e Função/Método.

4.4.1 Recurso Humano

Os recursos humanos são as pessoas que realizam as determinadas tarefas na área de manutenção de *software*. Cada pessoa possui suas respectivas características e pode atuar em uma atividade completa ou em uma etapa de uma atividade. Os principais campos que compõem essa entidade são:

- Nome: o nome completo do recurso;
- Imagem: a figura que representa o recurso;
- Atividades: uma ou mais atividades em que o recurso pode ser alocado;
- Etapas: uma ou mais etapas em que o recurso pode ser alocado;
- Tarefas: tarefas em que o recurso foi alocado.

4.4.2 Atividade

Conforme destacado na Seção 2.4.3, a área de manutenção de *software* possui diferentes atividades que não são constantes ou similares, dessa forma, essa entidade representa a pluralidade de atividades e o detalhamento de cada uma delas, possuindo como principais campos:

- Descrição: detalha a atividade que a ser executada;
- Tipo: determina o tipo da atividade na área de manutenção de *software*, conforme trazido na Seção 2.4.3, sendo Reparação de defeitos, Remoção de Módulos Falhos, Suporte a Usuários, Migração entre Plataformas, Conversão de Arquitetura, Adaptações Obrigatórias, Otimitização de Performance e Melhorias;
- Etapas: uma ou mais etapas que compõem a execução daquela atividade.

4.4.3 Etapa

Toda atividade possui as etapas que devem ser realizadas durante sua execução. O detalhamento de cada etapa visa determinar a quantidade de entrada e saída de artefatos, bem como a média de tempo para sua conclusão, dessa forma, seus principais campos são:

- Descrição: detalha a etapa a ser executada;
- Tipo: determina o tipo de etapa conforme a estrutura interna da organização que é realizada;

4.4.4 Tarefa

Toda e qualquer atividade é composta de tarefas, que são a materialização das solicitações de clientes, necessidades de mudanças e demais ações da área de manutenção. Dessa forma, o registro dessas tarefas está ligado ao planejamento das tarefas a serem realizadas e nas sugestões de recursos humanos a serem nela alocados. Logo, seus principais campos são:

- Descrição: detalha a tarefa a ser realizada;
- Tipo: determina o tipo da atividade na área de manutenção de *software* sendo Reparação de defeitos, Remoção de Módulos Falhos, Suporte a Usuários, Migração entre Plataformas, Conversão de Arquitetura, Adaptações obrigatórias, Otimitização de performance e Melhorias;
- Gravidade: determina o fator de impacto da tarefa, sendo Sem Gravidade, Pouco Grave, Grave, Muito Grave e Extremamente Grave;
- Urgência: determina o fator de tempo da tarefa, sendo Sem urgência, Pouco Urgente, Urgente, Muito Urgente e Extremamente Urgente;
- Tendência: determina a tendência de evolução da tarefa, sendo Sem tendência de piora, Piorar a longo prazo, Piorar a médio prazo, Piorar a curto prazo e Agravar rapidamente;
- Esforço previsto: esforço em horas previsto pela gerência;
- Dificuldade prevista: dificuldade prevista, em uma escala de 0 a 10, pela gerência, sendo 0 uma tarefa de menor dificuldade e 10 uma tarefa de maior dificuldade;
- Recursos: um ou mais recursos humanos alocados para a realização da tarefa.

4.4.5 Código-Fonte

Os códigos fontes, ou produtos, são a estrutura base do desenvolvimento de *software*, sendo que, para o *framework*, são os objetivos da realização da tarefa. Com isso, os principais campos dessa entidade são:

- Nome: nome do arquivo de código-fonte;
- Tarefas: uma ou mais tarefas que realizam ações no código-fonte.

4.4.6 Função/Método

As funções ou métodos são elementos que representam uma chamada de um determinado procedimento, ou seja, representam toda e qualquer ação ou execução que o *software* terá. Para o *framework*, elas consistem na parte fundamental das métricas de produto e é a partir delas que as bases de cálculo são realizadas. Assim sendo, seus principais campos são:

- Identificador: identificador da função ou método;
- Quantidade de Parâmetros: quantidade de entrada de parâmetros;
- Linhas de Código: quantidade total de linhas de código;
- Complexidade: complexidade ciclomática da função/método.

4.4.7 Métrica

A entidade métrica representa as informações utilizadas no cálculo do Grau da Perspectiva e Grau da Tarefa, bem como na determinação da alocação de recursos para composição do *framework*. Essa entidade controla os valores de cada métrica, sua perspectiva e a associação com as demais entidades. Seus principais campos são:

- Perspectiva: indica a perspectiva na qual a métrica está associada;
- Identificador: identificação da métrica;
- Valor: informação referente ao valor calculado para aquela métrica;
- Associação: relação da métrica com as demais entidades;

4.4.8 Demais entidades

Além das principais entidades mencionadas no decorrer das seções anteriores, o *framework* conta com algumas subentidades que auxiliam na produção de informações necessárias para a gerência das entidades e os cálculos das métricas.

4.4.8.1 Detalhamento

O detalhamento representa o grau de importância que a organização aplica às perspectivas de manutenção. Inicialmente, os percentuais são ajustados em 34% para Produto, 33% para Processo e 33% para Recursos, visando à avaliação igualitária destacada na Seção 4.1, que é melhor detalhada na Seção 4.5.1, contudo, as organizações podem ajustar conforme suas necessidades.

4.4.8.2 Usuário

Todos os registros de informações são atrelados a um usuário, cadastrado por meio de ferramenta de autenticação social. Os campos para armazenamento são:

- Email: o e-mail informado na criação da conta ou recebido através da ferramenta de autenticação;

- Nome: o nome informado na criação da conta ou recebido através da ferramenta de autenticação;
- Imagem: a foto do usuário informada na criação da conta ou recebida através da ferramenta de autenticação.

4.4.8.3 Projeto

Representa as informações gerais do projeto amplamente utilizadas nas métricas, tais como Conformidade com os padrões de engenharia de *software* e Índice de Manutenibilidade.

Essa entidade representa os diversos projetos que podem estar sob a responsabilidade do gestor, permitindo que ele gerencie toda a sua equipe de desenvolvimento, separando as responsabilidades por projeto.

A visualização da ferramenta é filtrada com base no projeto associado, e as demais informações são apresentadas de acordo com o projeto ao qual estão designadas.

4.4.8.4 Tipos Etapas

São os detalhamentos dos tipos de etapa que são específicos para as organizações, permitindo realizar uma segregação de recursos humanos para cada etapa da atividade.

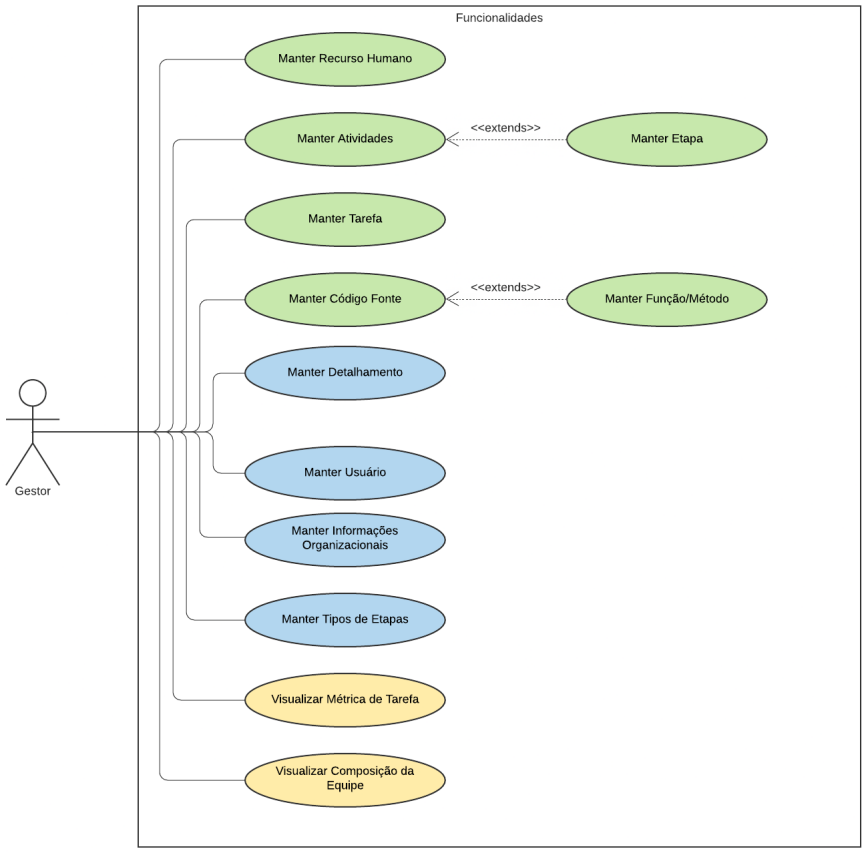
Por fim, como destacado na Seção 2.4.3, devido à variabilidade das tarefas presentes na área de manutenção de *software*, as etapas também podem variar. Portanto, essa entidade tem o objetivo de categorizar cada uma dessas etapas conforme o controle aplicado em cada organização.

4.5 FUNCIONALIDADES DA FERRAMENTA

De maneira a trazer maior solidez ao *framework*, as funcionalidades mapeadas estão pautadas na análise das três perspectivas de manutenção de *software* (produto, processo e recurso), conforme apontado na fundamentação teórica no Capítulo 2. Além disso, outro aspecto considerado são os diferentes tipos de atividades, que permitem uma abordagem mais ampla da ferramenta.

Para melhor descrever as principais funcionalidades, foi elaborado o diagrama de caso de uso, exibido na Figura 16. Destaca-se, porém, que como o intuito do *framework* é ser utilizado pelos gestores, existe apenas um perfil de usuário, denominado Gestor, com acesso a todas as funcionalidades do sistema.

Figura 16 – Diagrama de caso de uso



Fonte: Elaborado pelo autor (2023).

Com isso, as funcionalidades planejadas para o *framework* são:

- Manter Recurso Humano: é possível incluir, alterar, excluir e visualizar a lista de recursos humanos cadastrados no *framework*, estes são os recursos alocados nas tarefas e que servem de base para o cálculo e a análise da composição de equipe para atuação na tarefa, bem como gerenciar suas respectivas métricas;
- Manter Atividades: é possível incluir, alterar, excluir e visualizar a lista de atividades de manutenção de *software* cadastradas no *framework*, que são executadas pela equipe de manutenção de *software*, de acordo com os tipos de atividades levantados durante a fundamentação teórica;
- Manter Etapa: é possível incluir, alterar, excluir e visualizar a lista de etapas das atividades cadastradas no *framework*, estas são as etapas executadas pela equipe de manutenção de *software*;
- Manter Tarefa: é possível incluir, alterar, excluir e visualizar a lista de tarefas cadastradas no *framework*, para que se possa descrever a realização das atividades de acordo com as solicitações de entrada;

- Manter Código-Fonte: é possível incluir, alterar, excluir e visualizar a lista de arquivos de código-fonte cadastrados no *framework*, de maneira a gerenciar os produtos de *software* e suas respectivas métricas;
- Manter Função/Método: é possível incluir, alterar, excluir e visualizar a lista de funções ou métodos de um determinado código-fonte cadastrados no *framework*, para gerenciar os produtos de *software* e suas respectivas métricas;
- Manter Detalhamento: é possível alterar o percentual de distribuição de cada uma das perspectivas, para ajustar a melhor composição de métricas para a organização;
- Manter Usuário: é possível alterar as informações de perfil de usuário, bem como realizar a exclusão da conta;
- Manter Informações Organizacionais: é possível alterar as informações dos projetos e suas respectivas métricas;
- Manter Tipos de Etapas: é possível criar, alterar, excluir e visualizar os tipos de etapas utilizados no *framework*, de maneira a customizar a composição de equipe;
- Visualizar Métrica de Tarefa: através dessa funcionalidade, é possível identificar o grau de criticidade de uma determinada tarefa pela composição de métricas levantadas na fundamentação teórica, permitindo ao gestor compreender a priorização dessa tarefa;
- Visualizar Composição da Equipe: através dessa funcionalidade, é possível visualizar a sugestão de composição ideal de time para a tarefa, de maneira a otimizar a entrega. São sugeridos os recursos com base nas atividades e etapas que estes executam e alocação em outras tarefas.

As funcionalidades descritas foram implementadas no componente denominado *backend*. Haja vista que a comunicação com o *framework* se dá através de API, existe a possibilidade de visualizar os resultados e as informações tanto no contexto do componente de *frontend*, quanto diretamente através da camada de integração, podendo responder a outros sistemas de gerenciamento utilizados pela organização.

4.5.1 Fluxo do Sistema

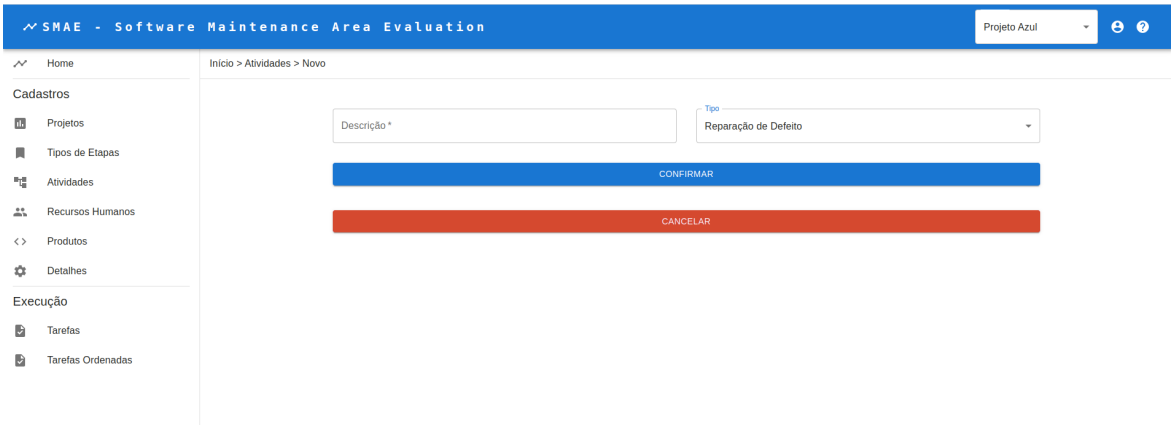
O ponto inicial do *framework* está no registro das informações básicas das perspectivas da manutenção de *software*, destacadas em verde nas Figuras 15 e 16, fornecendo as informações necessárias para os cálculos das métricas de cada perspectiva, conforme detalhado na seção 4.2.1, e composições finais dos resultados. Essas informações são inseridas manualmente pelo usuário na implantação da ferramenta.

Após, é necessário realizar o registro das informações complementares, destacadas em azul nas Figuras 15 e 16, de maneira a fornecer subsídios suplementares aos cálculos das métricas, bem como detalhar as informações de uso. Todas as telas estão dispostas no Apêndice E.

4.5.1.1 Criar Atividades — Perspectiva de Processo

Para a perspectiva de processo, são consideradas as atividades realizadas pela organização para atender às atividades de manutenção de *software*. Nesse caso, são registrados as atividades vinculadas à manutenção de *software* com as suas respectivas etapas. A Figura 17 representa esse cadastro.

Figura 17 – Cadastro de Atividades

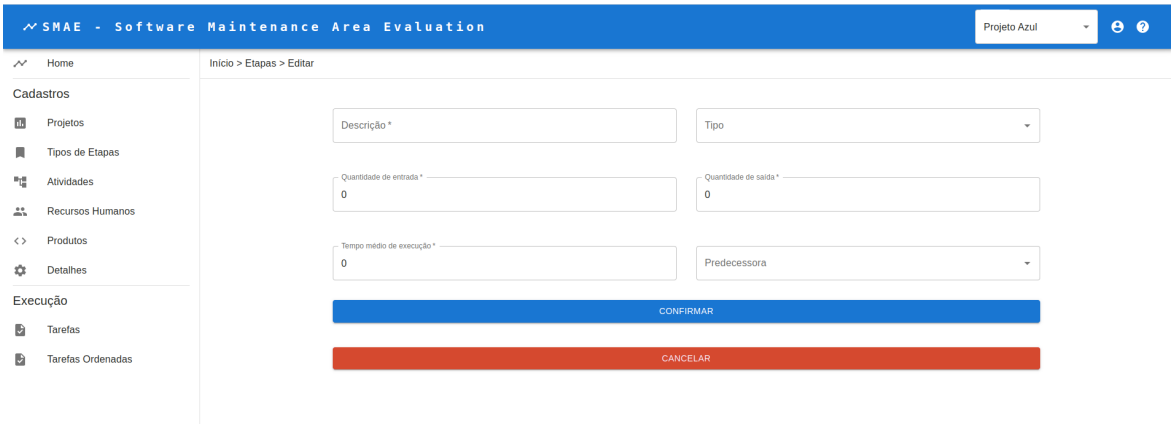


Fonte: Elaborado pelo autor (2023).

O detalhamento das atividades se dá com base em sua descrição e seu tipo, com as opções sugeridas conforme destacam Wazlawick (2013) e Jones (1998) e abordado na Seção 2.4.3[7], bem como é possível identificar as etapas de cada atividade.

A Figura 18 representa a criação de uma nova etapa na atividade.

Figura 18 – Cadastro de Etapas



Fonte: Elaborado pelo autor (2023).

[7] Os tipos sugeridos são Reparação de defeitos, Remoção de Módulos Falhos, Suporte a Usuários, Migração entre Plataformas, Conversão de Arquitetura, Adaptações Obrigatórias, Otimização de Performance e Melhorias.

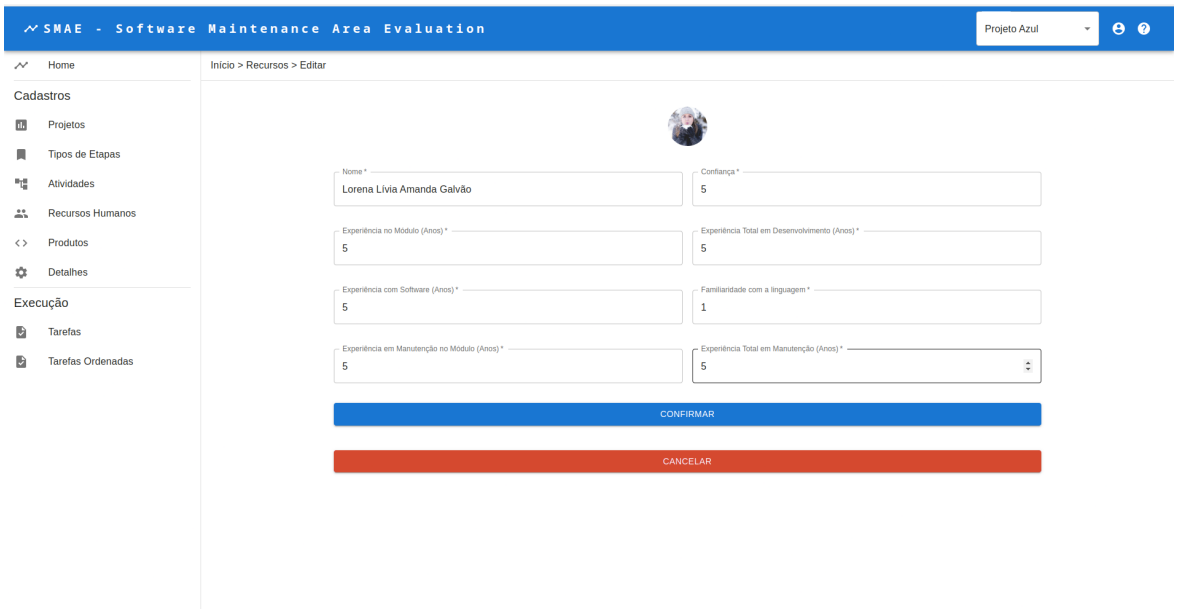
O detalhamento das etapas também se baseia em sua descrição e natureza, no entanto, os tipos de etapas são adaptados pela organização para proporcionar flexibilidade, em conformidade com as orientações de Varga (2018) e Ghanem (2020). Esses autores ressaltam que as atividades podem ser ajustadas de acordo com o contexto em que estão inseridas.

4.5.1.2 Criar Recursos Humanos — Perspectiva de Recurso

Para a perspectiva de recursos, a análise se concentra nos elementos humanos que podem ser designados para as diversas fases das atividades envolvidas na manutenção de *software*. Esses indivíduos desempenham um papel fundamental ao executar as tarefas necessárias para manter o *software* operante e funcional.

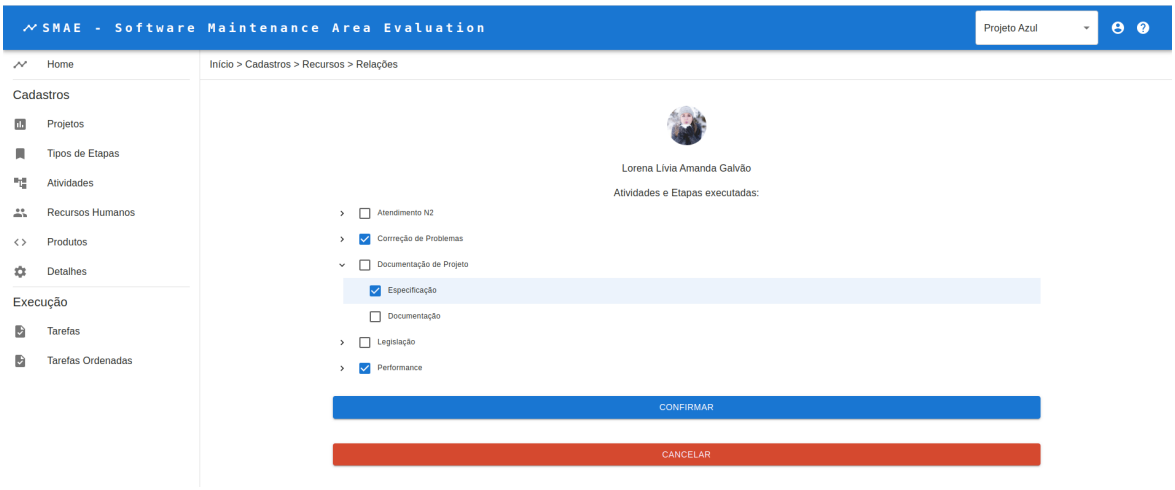
Também, são esses os recursos considerados para a composição de equipes, de maneira a sugerir a execução de uma determinada tarefa. Assim, ao explorar a perspectiva dos recursos, busca-se otimizar a gestão de recursos humanos, garantindo que os profissionais certos estejam envolvidos nas tarefas adequadas para manter o *software* em seu melhor estado e alcançar os objetivos de manutenção de maneira eficiente e eficaz. As Figuras 19 e 20 representam esse cadastro.

Figura 19 – Alteração de Recursos



Fonte: Elaborado pelo autor (2023).

Figura 20 – Atividades Realizadas pelo Recurso



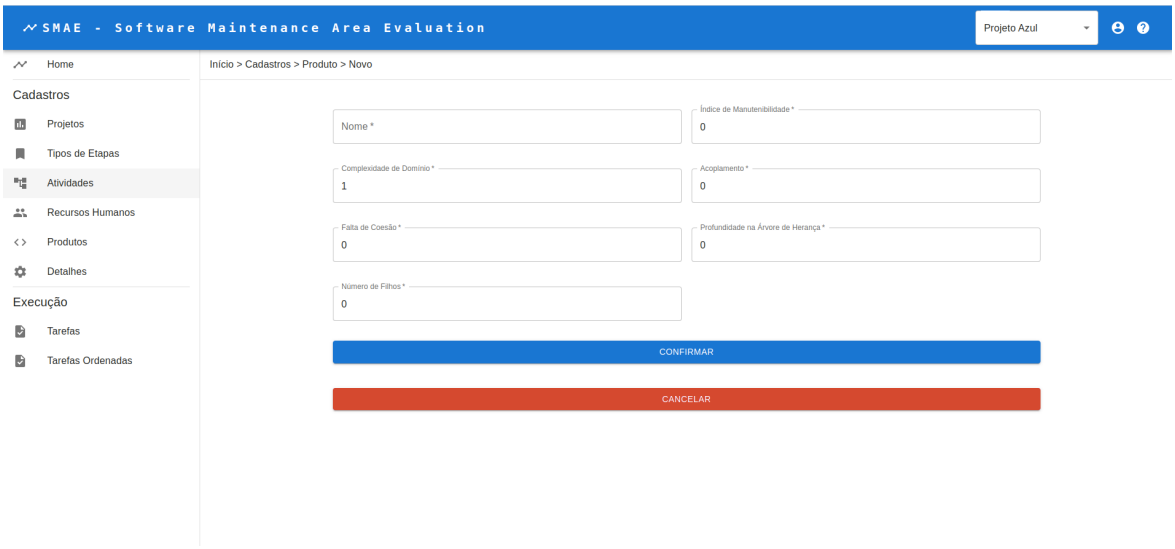
Fonte: Elaborado pelo autor (2023).

Para detalhar esses recursos, é necessário determinar um nome que serve para a identificação para o mesmo, podendo associá-lo a uma imagem (fotografia) que visa diferenciá-lo. Ainda, podem ser detalhadas quais atividades são realizadas por aquele recurso ou ainda quais etapas específicas de uma determinada atividade são executadas por aquele recurso.

4.5.1.3 Criar Código-Fonte — Perspectiva de Produto

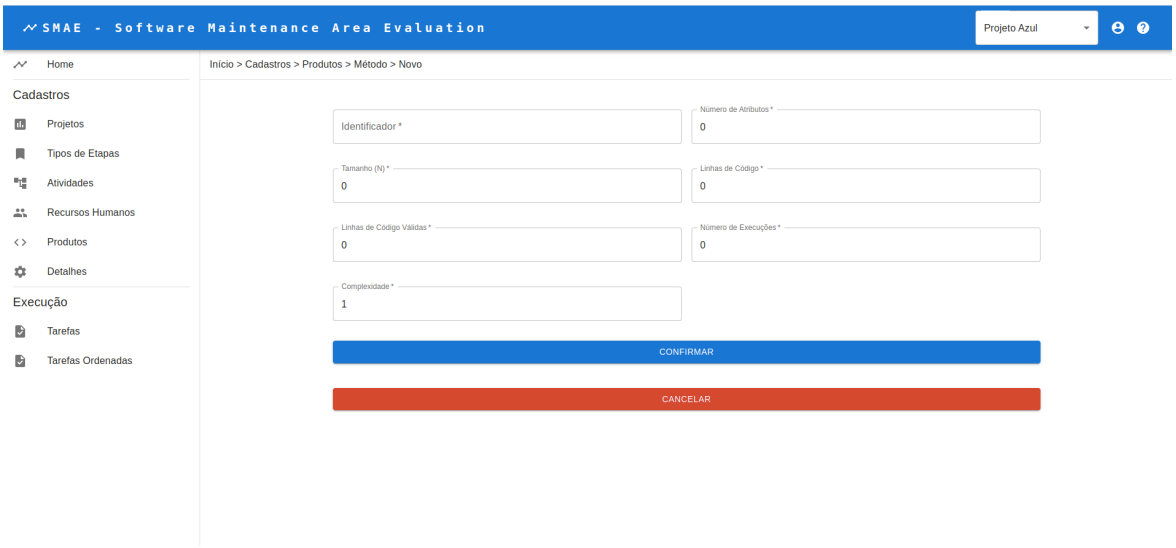
Para a perspectiva de produto, são considerados os códigos-fonte empregados na criação do *software*. Esses códigos contêm informações que descrevem as funcionalidades e/ou métodos utilizados para construir o produto. As Figuras 21 e 22 representam esse cadastro.

Figura 21 – Cadastro de Código-Fonte



Fonte: Elaborado pelo autor (2023).

Figura 22 – Cadastro de Funções/Métodos



Fonte: Elaborado pelo autor (2023).

Os identificadores dos códigos-fontes são seus respectivos nomes de arquivo, registrados da mesma forma no qual as organizações já os controlam atualmente.

Ainda, as funções e métodos são registradas para cada código-fonte, onde é possível detalhar as informações para os cálculos das métricas de produto, bem como realizar um mapeamento das funções a serem mantidas nas tarefas.

4.5.1.4 Informações complementares

Algumas informações que não estão diretamente ligadas às perspectivas de manutenção de *software* são utilizadas nos cálculos das métricas de manutenção de *software*.

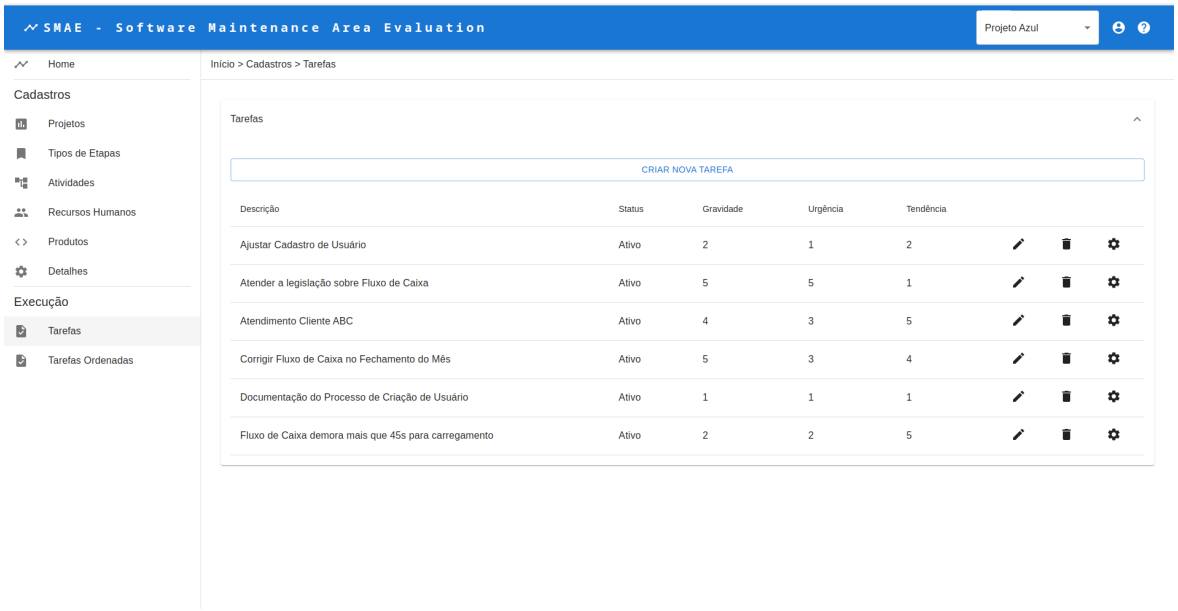
Desse modo, é criado um registro dessas informações de nível organizacional, denominado projeto, que serve de base para o cálculo das métricas. O intuito desse registro, que representa o caso de uso “Manter Informações Organizacionais”, é justamente fornecer valores absolutos utilizados no cálculo de maneira direta, sem intervenção das perspectivas.

4.5.2 Visualização de Resultados

Após os registros das informações sobre as perspectivas e informações complementares, o gestor realiza o registro das tarefas desenvolvidas durante a manutenção de *software*.

O registro de tarefa conta com as informações gerenciais da mesma, permitindo identificar uma breve descrição, o tipo de atividade de manutenção que é realizado, um esforço previsto inicial pelo gestor para a atividade, um grau de dificuldade prevista e uma matriz de Gravidade, Urgência e Tendência, que serve de informação adicional a ser representada ao final do cálculo, conforme observado na Figura 23.

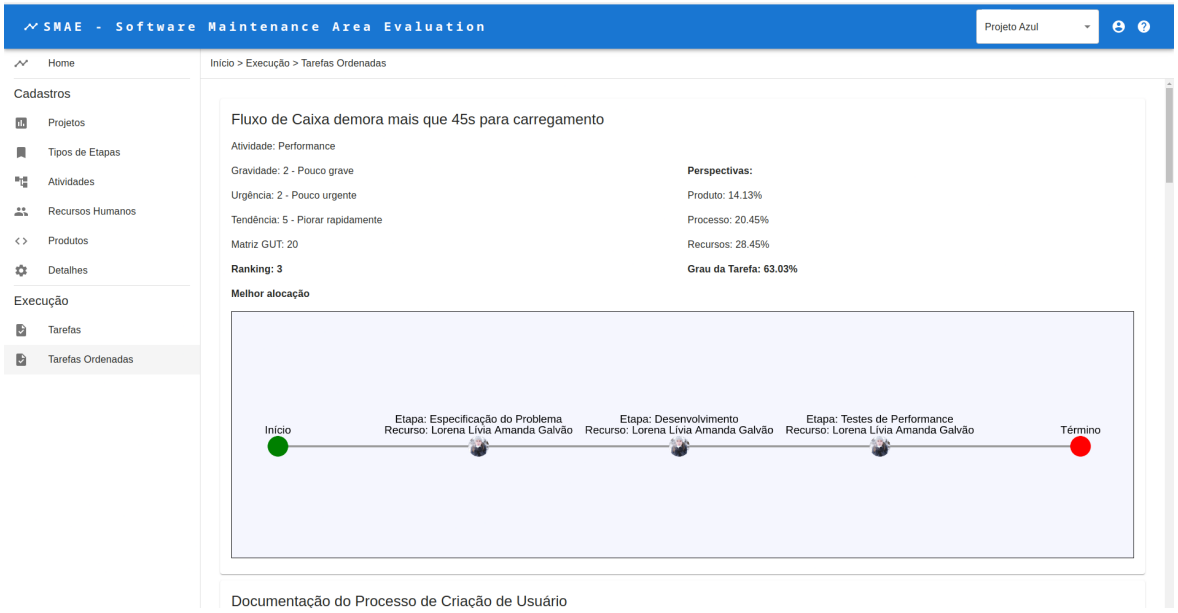
Figura 23 – Visualizar Tarefa



Fonte: Elaborado pelo autor (2023).

De maneira a identificar o grau de importância de cada tarefa, o gestor pode visualizar o Grau de Perspectiva e o Grau da Tarefa calculado com base nas métricas do *framework*, conforme exemplo exibido na Figura 24, em que a ordem das atividades é do maior Grau de Tarefa para o menor. Para a comparação do gestor, é demonstrado a ordenação pela matriz de Gravidade, Urgência e Tendência (Gravidade x Urgência x Tendência), de maneira a representar o conhecimento do gestor.

Figura 24 – Tarefa Ordenadas



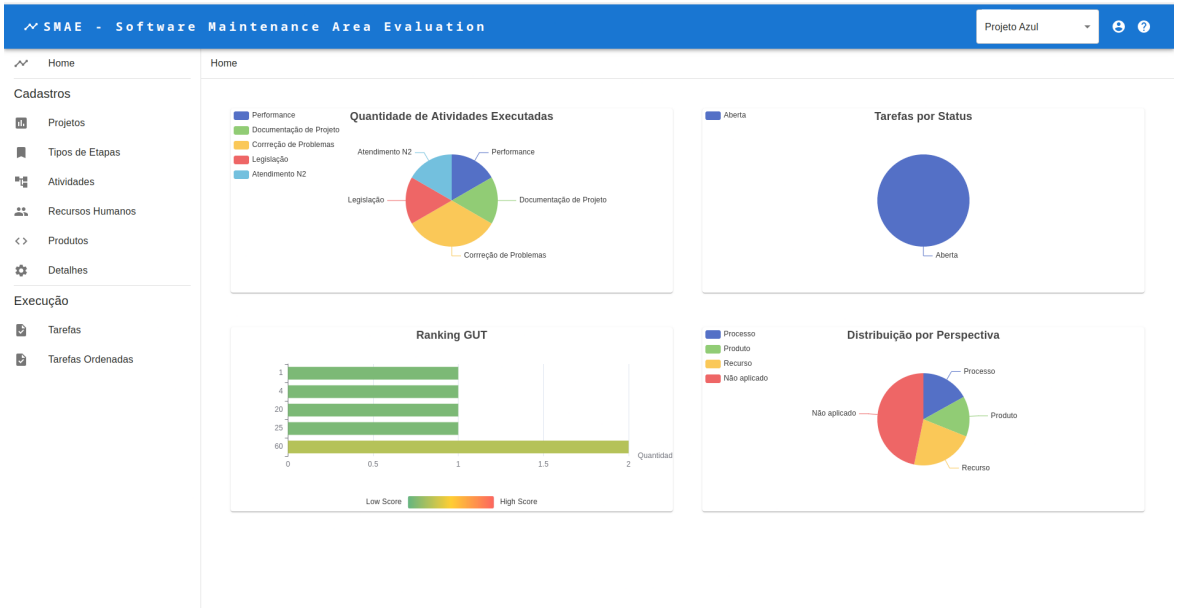
Fonte: Elaborado pelo autor (2023).

Também, é possível identificar os principais recursos e sugestão de recursos que podem

ser utilizados para realizar a tarefa registrada. A exibição dos recursos é feita com base na ordem de realização das etapas, nome do recurso e imagem associada a esse recurso.

Ainda, a ferramenta conta com uma tela de *dashboard*, que permite a avaliação do gestor sobre as tarefas já lançadas de uma forma mais abrangente, dando visibilidade as suas evoluções, como mostra a Figura 25.

Figura 25 – *Dashboard* Inicial



Fonte: Elaborado pelo autor (2023).

Para tanto, é exibido um gráfico de Quantidade de Atividades Executadas, que visa identificar quais as principais atividades realizadas pela equipe, as Tarefas por Status, que determina o andamento das atividades do gestor, o Ranking GUT, que visa demonstrar a quantidade de tarefas por priorização do gestor, e a Distribuição por Perspectiva, que visa demonstrar o percentual de utilização do framework em cada perspectiva.

4.6 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo, foram apresentadas as estruturas do *framework* que possibilitam aos gestores realizarem as avaliações das atividades de manutenção de *software*.

Todas as funcionalidades foram elaboradas de maneira a considerar as diferentes perspectivas de manutenção de *software*, que é uma das deficiências levantadas na fundamentação teórica e nos trabalhos relacionados, haja vista que a grande maioria dos *frameworks* e ferramentas apenas considera a perspectiva de produto, conforme levantado na Revisão Sistemática da Literatura, na Pesquisa Exploratória e também destacado por Meidan et al. (2018).

Outra característica importante do *framework* está na rápida disponibilidade em *Cloud*, garantindo a extensão de suas funcionalidades a outras ferramentas já comuns aos gestores e permitindo sua reutilização em outros sistemas, isso se dá por conta da utilização de APIs dispo-

nibilizadas pela ferramenta. Ainda, foi disponibilizada uma visualização padrão do *framework*, o que garante aos gestores também um fácil acesso quanto à visualização das informações.

Cabe ainda destacar que a apresentação de uma sugestão para a composição de equipes, destinada a atuar em determinada atividade, representa um diferencial para a ferramenta. Tal abordagem visa oferecer apoio ao gestor na tomada de decisões relativas à alocação de recursos, visando assegurar uma configuração adequada para todas as tarefas que serão executadas pela equipe de manutenção de *software*.

Como já destacado inicialmente, as métricas tiveram foco no paradigma de orientação a objeto, logo, o *framework* está direcionado para diagnosticar as atividades da área de manutenção de *software* que considerem métricas com essa abordagem de desenvolvimento. Além disso, pela abordagem de avaliação dos recursos ser abordada separada por projeto, conflitos de alocações podem ocorrer.

Além disso, foi identificada uma otimização para reduzir o tempo de implantação do *framework*. Isso envolve a leitura dos códigos-fonte e integrações com plataformas de gerenciamento de código, bem como com sistemas de gerenciamento de projetos, a fim de automatizar a coleta de métricas de produto, de processo e de recursos. Entretanto, para serem implantadas, se faz necessário um grande esforço técnico, haja vista a existência da pluralidade de ferramentas de mercado para controle de código-fontes e de gestão de processos e projeto, necessitando uma análise específica sobre como realizar a integração com cada uma dessas ferramentas. Logo, a ferramenta proposta permite que futuros trabalhos atuem na otimização da implantação, permitindo que o *framework* possua uma rápida aderência ao mercado e às organizações através da utilização de suas APIs.

5 EXPERIMENTO

O objetivo desta pesquisa foi implementar um *framework* para apoiar os gestores de manutenção de *software* na decisão sobre o atendimento e a priorização das tarefas e a composição da equipe destas tarefas de manutenção de *software*.

Como primeira etapa da pesquisa, a identificação do problema e a definição dos objetivos foi fundamentada no Capítulo 2 e 3. Na sequência, o Capítulo 4 apresentou o projeto e o desenvolvimento do *framework*, de maneira a solucionar as lacunas identificadas. Neste capítulo, são apresentadas as etapas de Demonstração e a Avaliação do processo metodológico do *Design Science Research Methodology (DSRM)*, pois o experimento visa demonstrar o uso do artefato na solução do problema e avaliar sua aplicação no contexto empresarial com seu uso pelos gestores da área de manutenção de *software*.

Como metodologia de avaliação, o *framework* proposto foi aplicado em ambientes reais, com a participação de cinco gestores, sendo coletados feedbacks por meio de uma entrevista semiestruturada. Além disso, foi utilizado um questionário *online* com base no modelo TAM (*Technology Acceptance Model*, ou Modelo de Aceitação Tecnológica) para coletar dados junto a uma base maior de gestores da área de manutenção de *software*, a fim de validar a aceitação do *framework*. Este capítulo apresenta os resultados obtidos nessa avaliação. Na seção 5.1, é apresentado o planejamento da avaliação do *framework*, na seção 5.2, são apresentados os detalhes referente às aplicações do *framework* e dos questionários, na seção 5.3, são apresentadas as análises referente à coleta de dados e, por fim, na seção 5.5, são apresentadas as considerações finais.

5.1 PLANEJAMENTO DA AVALIAÇÃO

A avaliação do artefato é fundamental para determinar sua eficácia e utilidade, podendo ser realizado por meio de experimentos, de simulações, de estudos de caso ou de outros métodos de avaliação adequados ao problema. Hevner et al. (2004) destacam o uso de experimentos controlados, em particular, como um meio viável para validar as qualidades (como usabilidade e aplicabilidade) de um artefato desenvolvido por intermédio do DSRM.

Logo, visando à experimentação, o *framework* foi aplicado por meio da ferramenta desenvolvida em um ambiente controlado, durante o período de dezembro de 2022 até janeiro de 2023, para esta avaliação. Para isso, foram elaborados dois roteiros de entrevistas semiestruturadas, detalhados na Seção 5.2.

A entrevista semiestruturada foi aplicada para a coleta de dados qualitativos, de maneira a conferir flexibilidade ao entrevistador, permitindo a adaptação das perguntas com base no contexto de avaliação, bem como trouxe profundidade e riqueza de dados.

Também, durante a primeira entrevista, foi realizada uma demonstração da ferramenta desenvolvida, como meio de treinamento dos gestores em sua utilização. Durante todo o processo, o pesquisador acompanhou diretamente os gestores, a fim de auxiliar em caso de dúvidas sobre a

utilização do sistema. A aplicação necessitou de um computador com um navegador (*browse*) atualizado e acesso à rede de internet.

Por fim, visando à avaliação quantitativa do *framework*, foi aplicado um questionário com base no modelo TAM, que tem o objetivo de identificar as aceitações de uma tecnologia por seus usuários, através da aplicação de um questionário.

5.1.1 Protocolos Aplicáveis ao Experimento

Para este experimento foram utilizados os seguintes protocolos:

- **Termo de Consentimento Livre e Esclarecido (TCLE)**: foi aplicado em todas as entrevistas, registrado por formulários online, bem como na aplicação do questionário TAM. Disponível no Apêndice A, é um documento que descreve os benefícios e os riscos da participação voluntária na pesquisa.
- **Entrevista Semiestruturada**: foi aplicada durante a experimentação do *framework* e permitiu ao entrevistador e ao entrevistado conduzirem os tópicos a serem abordados conforme as necessidades de avaliação. Sua estrutura é detalhada na seção 5.2.
- **Questionário TAM**: foi aplicado para a avaliação dos fatores de aceitação do *framework*. Disponível no Apêndice D, sua estrutura é detalhada na seção 5.2.1.

5.1.2 Estrutura do Experimento

Com o intuito de responder à questão de pesquisa desta investigação, se faz necessário a comparação entre o estado atual do planejamento das tarefas de manutenção de *software* e o estado após a utilização da metodologia proposta pelo *framework*.

O experimento por meio da seleção por conveniência dos gestores avaliou a aceitação de uso da ferramenta no planejamento das tarefas de manutenção de *software*.

Os grupos participantes utilizaram o *framework* durante a prática da gestão de suas áreas de manutenção de *software*, aplicando as sugestões fornecidas pela ferramenta. Dessa forma, cada participante pode identificar benefícios e fragilidades que o *framework* possui, com o objetivo principal de avaliar se o *framework* atende as suas necessidades.

Pela utilização de uma ferramenta, esta pesquisa é caracterizada como experimental, pois sistematicamente provocará alterações no ambiente a ser pesquisado, no qual será validado se a intervenção produzirá os efeitos desejados (WAZLAWICK, 2014).

O questionário TAM foi direcionado através de envio de formulário diretamente para os participantes (por conveniência), diferentemente do grupo de participantes que utilizaram o *framework*, que possuíam algum vínculo conhecido com a área de manutenção de *software* e também publicado em mídias sociais, tais como LinkedIn e Facebook, em grupos específicos sobre o assunto de manutenção de *software*.

5.2 APLICAÇÃO DO EXPERIMENTO

O *framework* foi aplicado com cinco gestores da área de manutenção de *software*, cada um deles com quantidades distintas de colaboradores sob sua supervisão, sendo representado pelo Quadro 17. A primeira entrevista teve uma duração média de 2 (duas) horas, por ser realizada a demonstração da ferramenta, bem como o roteiro inicial previa questões de caracterização dos gestores.

Quadro 17 – Equipes dos grupos

Gestor	**Nº Pessoas**
A	6
B	4
C	4
D	6
E	8

Fonte: Elaborado pelo autor (2023).

Como o primeiro roteiro teve o intuito de caracterizar o grupo de aplicação, suas perguntas foram direcionadas à identificação e à categorização dos participantes, cujas questões são detalhadas no Quadro 18. Sua aplicação ocorreu na primeira entrevista com os participantes, realizada entre 16 de dezembro de 2022 e 23 de dezembro de 2023. As perguntas elaboradas foram:

Quadro 18 – Questões e Motivações do Roteiro Inicial

(continua)

Categoria	**Questão**	**Motivação**
Pessoal	Qual a sua idade?	Identificar a correlação com a idade de cada participante
Pessoal	Qual a sua cidade/região?	Identificar a região demográfica da aplicação
Pessoal	Quando foi o seu primeiro contato com a área de manutenção de *software*?	Identificar a maturidade de conhecimento da área
Pessoal	Há quanto tempo você realiza a gestão de uma área de manutenção de *software*?	Identificar a maturidade de gestão da área
Pessoal	Atualmente, qual a nomenclatura do seu cargo?	Identificar os principais cargos envolvidos

Fonte: Elaborado pelo autor (2023).

Quadro 18 — Questões e Motivações do Roteiro Inicial

(conclusão)

Pessoal	Qual o seu grau de instrução?	Identificar a correlação com a instrução de cada participante
Empresa	Descreva quais as principais atividades trabalhadas no processo de manutenção de *software* atualmente.	Identificação das atividades em que o *framework* será aplicado
Empresa	Descreva quais as principais etapas das atividades detalhadas acima.	Identificação das etapas em que o *framework* será aplicado
Empresa	A empresa entende a área de manutenção de *software* como algo estratégico?	Identificação do nível estratégico da empresa com relação à área de manutenção de *software*
Empresa	Qual o custo médio mensal (percentual) de gastos com a área de manutenção de *software* na equipe?	Identificação da importância da área de manutenção de *software* para a empresa
Empresa	Quais os instrumentos (ou ferramentas) utilizados para conduzir a gestão da equipe de manutenção de *software*? Há alguma ferramenta computacional que auxilie a gestão?	Identificação de ferramentas utilizadas pela gestão
Empresa	Você acredita que uma ferramenta pode auxiliar o gestor na tomada de decisão em relação à priorização das tarefas e alocação da equipe? Como?	Identificação da intenção de uso de uma ferramenta com a mesma abordagem proposta pelo *framework*

Fonte: Elaborado pelo autor (2023).

Foram categorizadas as perguntas de identificação em "Pessoal", nas quais está relacionada a caracterização do perfil dos gestores, e "Empresa", nas quais está relacionada a caracterização do ambiente empresarial no qual o gestor está inserido.

Além disso, ao longo do período de aplicação, eram feitas quinzenalmente entrevistas com a aplicação de outra entrevista semiestruturada, para identificar a aplicação do *framework*. Para essa entrevista, foram aplicadas as questões detalhadas no Quadro 19.

Quadro 19 – Questões e Motivações do Roteiro de Acompanhamento

(continua)

Categoria	**Questão**	**Motivação**
Ferramenta	Sobre a separação das tarefas, das atividades, dos produtos e dos recursos em projetos, atendeu a sua estrutura de gestão? Detalhe.	Identificação do entendimento de cada perspectiva da área de manutenção de *software*
Ferramenta	Sobre o controle das atividades e respectivas etapas, atenderam a sua estrutura de gestão? Detalhe.	Identificação da aplicação da perspectiva de Processo
Ferramenta	Sobre o controle dos produtos e respectivos métodos, atenderam a sua estrutura de gestão? Detalhe.	Identificação da aplicação da perspectiva de Produto
Ferramenta	Sobre o controle dos recursos humanos e alocação prévia nas etapas, atenderam a sua estrutura de gestão? Detalhe.	Identificação da aplicação da perspectiva de Recursos
Ferramenta	Sobre o cadastro de tarefas, atendeu a sua estrutura de gestão? Detalhe	Identificação da completude do controle de tarefas
Ferramenta	A visualização da priorização de tarefas, com as informações sobre os motivos e detalhamentos, ficou clara?	Identificação da disposição da sugestão de priorização
Ferramenta	A visualização sobre a alocação de recurso sugerida, ficou clara?	Identificação da disposição de sugestão de alocação
Ferramenta	Há algum processo ou informação que você considera faltante no projeto desenvolvido que lhe auxiliaria na tomada de decisão?	Identificação da falta de informação relevante para a tomada de decisão

Fonte: Elaborado pelo autor (2023).

Quadro 19 — Questões e Motivações do Roteiro de Acompanhamento

(conclusão)

Percepções	Há percepção de alterações na tomada de decisão e/ou alocação de equipe após a utilização do *framework*? Se sim, qual?	Identificação da percepção de mudança na tomada de decisão
Percepções	Houve alguma necessidade de intervenção nas sugestões apontadas pelo *framework*?	Identificação da assertividade nas sugestões do *framework*
Percepções	Há alguma similaridade entre o processo de alocação atual e as sugestões trazidas pelo *framework*?	Identificação de similaridade com o processo atual de alocação
Percepções	Percebeu-se que o *framework* auxilia na tomada de decisão, dando suporte à escolha da priorização de tarefas e/ou alocação de recursos?	Identificação de importância e uso do *framework*
Percepções	Há confiança nas sugestões apontadas pelo framework? Há maturidade nas sugestões apontadas pelo *framework*?	Identificação de maturidade das sugestões do *framework*
Percepções	Há alguma sugestão de melhoria para que o *framework* possa se adequar mais ao processo de manutenção de *software* ou atender a algum processo de manutenção de *software* faltante?	Identificação de eventuais melhorias
Outros	Há alguma consideração a ser feita além das ponderações já realizadas? Se sim, descreva abaixo.	Identificação de sugestões não abordadas nas demais questões

Fonte: Elaborado pelo autor (2023).

As entrevistas tiveram duração aproximada de 1 (uma) hora e as perguntas foram categorizadas em três grupos distintos:

- **Ferramenta**: as perguntas desta categoria são relacionadas a identificação do uso da ferramenta, disposições visuais e demonstração das informações;
- **Percepção**: as perguntas desta categoria são relacionadas à percepção dos gestores e à identificação de como o *framework*, por meio da ferramenta, afetou o planejamento das tarefas de manutenção de *software*;
- **Outros**: as perguntas desta categoria são de cunho geral.

Ambos os roteiros para a condução das entrevistas foram implementados utilizando o *Google Formulários*, sendo estruturados da seguinte maneira:

- Roteiro Inicial:
  - Seção 1: Termo de Consentimento Livre e Esclarecido - TCLE;
  - Seção 2: questões demográficas e para identificação do perfil do gestor;
  - Seção 3: questões para identificação das características da empresa e área de manutenção de *software*;
  - Seção 4: agradecimentos.
- Roteiro de Acompanhamento:
  - Seção 1: Termo de Consentimento Livre e Esclarecido - TCLE;
  - Seção 2: questões sobre a Ferramenta;
  - Seção 3: questões sobre a percepção dos gestores sobre o *framework*;
  - Seção 4: demais questões;
  - Seção 5: agradecimentos.

Todos os gestores utilizaram o *framework* através da ferramenta proposta, necessitando de aproximadamente dois dias úteis (cerca de 14 horas) para realização dos cadastros básicos e lançamentos de informações iniciais na ferramenta. O uso do *framework* se deu após os cadastros iniciais.

5.2.1 Aplicação do TAM

Ao final do experimento, foi elaborado um questionário TAM, disponível no Apêndice D, para avaliação da aceitação do *framework* por parte dos usuários, sendo aplicado por meio de um formulário *online* com um maior público de gestores.

Esse questionário foi baseado no modelo TAM (*Tecnology Acceptance Model*), que foi originalmente proposto por Davis (1989). Ademais, Aguiar, Kemczinski e Gasparini (2017), através de sua pesquisa, identificaram que se trata de um modelo válido e robusto, amplamente

utilizado no meio acadêmico e profissional, bem como é aderente à avaliação de *software* para gestão de projetos.

Conforme demonstrado por Aguiar, Kemczinski e Gasparini (2017), o questionário original, proposto por Davis (1989), é composto por nove perguntas dispostas em três grupos de perguntas, fornecendo opções de respostas no padrão de uma escala Likert, conforme demonstra o Quadro 20.

Quadro 20 – Questionário TAM Base

Grupo	**Questão**
Declarações de utilidade percebida	1. Usar a tecnologia melhoraria meu desempenho ao fazer meu trabalho? 2. Usar a tecnologia no trabalho melhoraria minha produtividade? 3. Usar a tecnologia aumentaria minha eficácia no meu trabalho? 4. Eu entendo que a tecnologia é útil no meu trabalho?
Declarações de facilidade de uso	5. Aprender a operar a tecnologia seria fácil para mim? 6. Eu irei encontrar de forma fácil as funcionalidades do sistema para obter as respostas necessárias? 7. Seria fácil para mim tornar-me hábil no uso da tecnologia? 8. Eu acharia a tecnologia fácil de usar?
Intenção comportamental de usar	9. Pretendo utilizar a tecnologia regularmente no trabalho?

Fonte: Adaptado de Aguiar, Kemczinski e Gasparini (2017).

Por meio dessa base, foi desenvolvido um questionário para o *framework* SMAE composto de três seções de perguntas. A primeira seção corresponde a informações de perfil e a segunda seção corresponde a informações profissionais, ambas usadas para a caracterização do respondente do questionário.

Por fim, a última seção corresponde às questões TAM elaboradas para o *framework*, conforme demonstra o Quadro 21, seguindo uma escala Likert de cinco pontos, sendo que para cada pontuação foi atribuído um valor de um até cinco, em que o valor de pontuação mais baixo inicia em um e o valor de pontuação mais alto termina em cinco. O valor mais baixo corresponde a discordo totalmente e o valor mais alto corresponde a concordo totalmente.

Quadro 21 – Questionário TAM para o SMAE

Grupo	**Questão**
Utilidade percebida	1. As demonstrações de uso foram suficientes para compreender as funcionalidades da ferramenta 2. Usar a ferramenta melhora o meu desempenho durante o planejamento das tarefas de manutenção de *software* 3. Usar a ferramenta aumenta a minha produtividade como gestor da área de manutenção de *software* 4. A ferramenta é útil para determinar a alocação de pessoas nas tarefas de manutenção de *software*
Facilidade de uso	5. Com a demonstração, foi fácil entender a ferramenta 6. Não há dificuldades em localizar as ações a serem realizadas na ferramenta 7. É fácil a utilização da ferramenta para meu apoio nas atividades de planejamento das tarefas de manutenção de *software* 8. A ordem de utilização da ferramenta está disposta de forma lógica, facilitando o uso
Intenção de uso	9. A disposição do Grau de Tarefa está de fácil localização e sua ordenação é útil 10. Eu utilizaria a ferramenta se disponível em minha rotina de trabalho 11. Minha organização aceitaria a utilização da ferramenta como ferramenta de trabalho

Fonte: Elaborado pelo autor (2023).

Por fim, foi incluída uma questão aberta, visando coletar as impressões, as sugestões de melhoria e os elogios dos participantes da pesquisa. O questionário foi implementado utilizando o *Google Formulários*, sendo estruturado da seguinte maneira:

- Seção 1: Termo de Consentimento Livre e Esclarecido - TCLE;
- Seção 2: questões demográficas e para identificação do perfil do gestor;
- Seção 3: questões sobre os 3 (três) constructos[1] do TAM;

[1] Utilidade percebida, facilidade de uso percebida e intenção de uso

- Seção 4: agradecimentos.

Ainda, durante a Seção 3 do questionário, foi anexado um vídeo[2], com duração de 8 minutos, contendo uma introdução sobre o *framework* e demonstrando as funcionalidades da ferramenta. O propósito foi dar subsídios aos participantes para procederem às respostas do questionário.

Esse questionário foi aplicado inicialmente como teste piloto com três especialistas da área de manutenção de *software*, tendo como finalidade identificar melhorias e ajustes antes da aplicação final com o público.

O tempo médio de respostas ao questionário foi de 15 minutos, por meio do qual o vídeo foi avaliado como claro, direto e suficiente para demonstração da proposta e da ferramenta. Foram apontadas correções básicas de ortografia e gramática, bem como a consideração do tempo total no qual o gestor está na área de manutenção de *software*.

O questionário foi então direcionado para a aplicação ao público e encontra-se em sua íntegra no Apêndice D.

5.3 ANÁLISE DOS DADOS

Foram realizadas um total de cinco entrevistas com cada gestor, totalizando 25 entrevistas, sendo uma inicial e quatro de acompanhamento. Ao todo, foram acompanhados aproximadamente dois meses de atividades dos gestores. Todas as entrevistas foram realizadas na modalidade *online*, através da ferramenta *Google Meet*.

Para analisar as entrevistas semiestruturadas, no contexto do *Design Science Research Methodology*, foi utilizada a análise de conteúdo. Por proporcionar uma avaliação qualitativa, esse método permitiu extrair padrões a partir do conteúdo textual das entrevistas, visando identificar temas recorrentes e *insights* relevantes.

Durante a análise, foram agrupados os dados, para organizar as respostas dos participantes de acordo com tópicos relacionados à avaliação do *framework*. Com isso, permitiu-se identificar percepções, experiências e opiniões dos entrevistados, contribuindo para a construção de conhecimento sobre a relevância do artefato proposto. Ainda, todos os entrevistados aceitaram o TCLE em todas as entrevistas.

Dessa forma, foram identificadas, nas questões de perfil de gestor, os seguintes contextos de aplicação:

- Idade: a idade média dos gestores foi de 28 anos;
- Região: todos os gestores respondentes são de Joinville/SC;
- Tempo na Área de Manutenção de *software*: o tempo médio na área de manutenção de *software* foi de 8 anos;

[2] Vídeo de apresentação do *framework* SMAE: https://youtu.be/3Palk3sui3A

- Tempo na Gestão da Área de Manutenção de *software*: o tempo médio na gestão da área de manutenção de *software* foi de cinco anos.
- Cargos: Coordenador e Gestor
- Escolaridade: Bacharelado e Pós-Graduação (*Lato Sensu*)

Os perfis de profissionais entrevistados foram detalhados a seguir, identificados com a letra G e um número sequencial, sendo que a ordem seguida foi a de realização das entrevistas:

- G1: possui formação em Engenharia da Computação, exerce o cargo de Coordenador de *software* há aproximadamente 4 anos. Atua principalmente com Bug, Melhoria e Atendimento.
- G2: possui formação em Engenharia de *software* e Pós-Graduação em Gestão de Projetos, exerce o cargo de Coordenador de *software* há aproximadamente 10 anos. Atua principalmente com Reparação de Defeitos, Adaptações Obrigatórias, Otimização de Performance e Melhorias.
- G3: posui formação em Análise e Desenvolvimento de *software*, exerce o cargo de Gestor de Sistemas há aproximadamente um ano. Atua principalmente com Customização e parametrização de processos, correção de eventuais erros ou orientação da utilização correta do *software*.
- G4: possui formação em Análise e Desenvolvimento de *software*, exerce o cargo de Coordenador há aproximadamente quatro anos. Atua principalmente com análise do problema, avaliação da prioridade e alocação de recurso para solução.
- G5: possui formação em Engenharia de *software* e Pós-Gaduação em Gestão de Projetos, exerce o cargo de Gestor de Sustentação há aproximadamente seis anos. Atua principalmente com Correção de Defeitos.

Os entrevistados mencionaram que o processo para implantação e utilização da ferramenta demandou um grande esforço e tempo, mas sinalizaram que isso resulta não apenas na melhoria dos próprios processos, mas também na área de manutenção. Outro ponto importante é que as visões dos processos de Reparação de Defeito, Adaptação Obrigatória, Otimização de Performance e Melhorias foram semelhantes para todos os entrevistados.

Ainda, todos os entrevistados apontaram que suas empresas entendem que a área de manutenção de *software* é essencial para a retenção de clientes e à adaptação do *software* ao mercado, contudo, demonstraram preocupações com relação à aceleração das atividades de Reparação de Defeitos, principalmente para atendimento de clientes relevantes. Apontaram que um alinhamento com a análise de processos é essencial para o sucesso da área.

Por fim, indicaram que os gastos com manutenção de *software* estão, em média, com cerca de 75% de sua área, bem como não há a aplicação de ferramentas computacionais, apenas de gestão, como JIRA, Azure, Trello e GLPI. Por este motivo, todos acreditam que uma ferramenta que auxilie o gestor na verificação de priorização, de grau de dificuldade e da alocação correta pode fazer com que os processos se desenvolvam de forma mais ágil e assertiva, ampliando a visão do gestor sobre os processos e potencializando os resultados ao longo prazo. A ferramenta pode auxiliar, também, na estruturação de planos de ação que reduzam os tempos de entrega, ajudando a identificar os pontos críticos no processo de manutenção.

5.3.1 Entrevistas de Acompanhamento

Nas entrevistas de acompanhamento, as questões aplicadas estão ligadas diretamente à categoria de Ferramenta, para avaliação do *framework* e ferramenta desenvolvidos, e Percepção, para avaliar a percepção dos gestores durante a utilização do *framework*.

Todos os gestores informaram que a separação em Atividades, Produtos, Recursos e Tarefas é clara e possui relação com a condução de suas atividades atuais, bem como a separação em projetos auxilia também no controle e na condução de projetos diferentes.

Ainda, sobre a perspectiva de Processo, o controle das atividades e das etapas também foi bem compreendido pelos gestores, sendo que quase em sua totalidade não houve necessidades de melhoria. Todavia, o gestor G2 sugeriu que poderia haver uma estrutura de detalhamento de cada etapa, o que facilitaria a compressão e a gestão compartilhada.

No que diz respeito à perspectiva de Produto, os gestores G1, G2 e G4 indicaram que, além do controle dos códigos-fonte, é necessário também o controle de outros produtos, como documentações e itens similares. Portanto, o atendimento a essa perspectiva no *framework* é parcial. Além disso, o gestor G4 enfatizou a necessidade de uma importação automática das estruturas de codificação, o que simplificaria uma futura implantação do *framework*.

Sobre a perspectiva de Recursos, todos trouxeram o benefício de controle da separação de alocação dos recursos na atividade ou em etapas específicas, facilitando a definição da gestão das atividades. Entretanto, pelo período de aplicação do instrumento, houve colaboradores afastados e em férias (recessos natalinos). Como o *framework* não possui um controle de indisponibilidade, acabou não atendendo em alguns cenários.

No controle de tarefas, a comparação com a Matriz GUT[3] foi apontada como essencial pelos gestores, por ser considerada uma referência na tomada de decisão. Os gestores G2 e G4 sugeriram como melhoria a integração com as plataformas de controle de projeto Azure, bem como o gestor G1 solicitou uma forma de priorização forçada pelo próprio gestor, sendo que o *framework* poderia se adequar a essa restrição.

No que se refere às sugestões de alocação, não foram identificadas ponderações para melhoria, visto que as informações foram consideradas claras e consistentes. No entanto, o gestor

[3] Gravidade, Urgência e Tendência.

G5 ressaltou a necessidade de apresentar alocações secundárias como um suporte adicional ao gestor, em situações de restrições com a alocação principal.

Nas avaliações referentes às percepções dos gestores, observou-se que houve modificações na tomada de decisões como resultado da utilização do *framework*. Todos os gestores relataram que tarefas que normalmente não seriam priorizadas passaram a ser consideradas relevantes devido às sugestões apresentadas. Inicialmente, os gestores também apontaram que as sugestões apresentadas pelo *framework* de alocação de recurso também foram ajustadas, contudo, no decorrer da aplicação não houve essa necessidade, apenas em situações que exigiram o processo (como ausências ou recessos).

Todos os gestores entenderam que as sugestões apontadas pelo *framework* trouxeram uma melhor dinâmica para a área de manutenção de *software*, bem como as priorizações proporcionaram uma aceleração nas realizações das tarefas, sendo considerado que o *framework* auxiliou na tomada de decisão.

Ainda, também em sua totalidade, os gestores acreditam haver confiança na utilização do *framework* assistidamente, sendo necessárias algumas intervenções pontuais.

Por fim, outro ponto de destaque apontado pelos gestores G1, G2 e G4 é que não identificaram outra ferramenta de mercado que tenha esse propósito, oportunizando uma nova aplicação para a área de manutenção de *software*. O pesquisador ainda constatou que, mesmo disponível, nenhum gestor utilizou a autenticação por meio do *Google Identity Platform*.

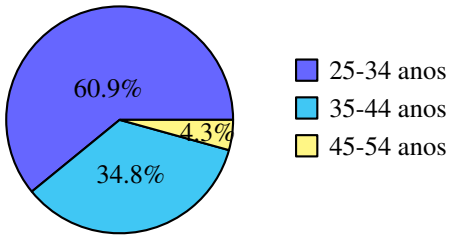
Diante disso, nos comentários coletados, todos os gestores consideraram a ferramenta como aplicável a área de manutenção de *software*, sendo que possui caráter de auxílio na tomada de decisão em priorização de tarefas e alocação de recursos. Algumas palavras entre os comentários que corroboram com essa análise são: "visão racional dos pontos", "aceleração de expedições", "similaridades no processo", "qualifica o processo atual", "resultado padrão"e "nenhuma ferramenta similar no mercado".

5.3.2 Questionário TAM

Após a aplicação do *framework* em ambiente real, foi aplicado um questionário TAM com um maior público de gestores, visando avaliar a aceitação da tecnologia. Sua divulgação foi feita por meio de mídias sociais e também por convites por conveniência. O questionário ficou ativo entre o período de 01 julho de 2023 até 15 de setembro de 2023.

Nesse período, obteve-se a participação de 23 respondentes. Na Figura 26, é apresentada a quantidade de respondentes por faixa etária, sendo que a maioria deles está entre 25 e 34 anos.

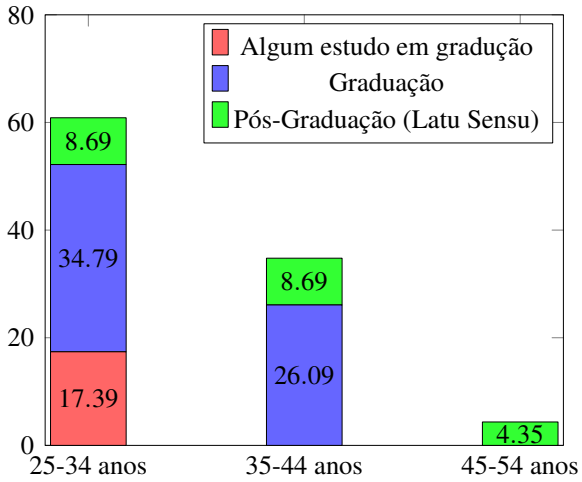
Figura 26 – Resultado TAM - Faixa Etária



Fonte: Elaborado pelo autor (2023).

Na Figura 27, é apresentada a relação entre o nível de escolaridade por faixa etária, sendo que o principal nível de escolaridade é Graduação, com 60,87% das respostas, indicando que o alcance da pesquisa foi a um público com um nível médio de escolaridade.

Figura 27 – Resultado TAM — Idade por Nível de Escolaridade



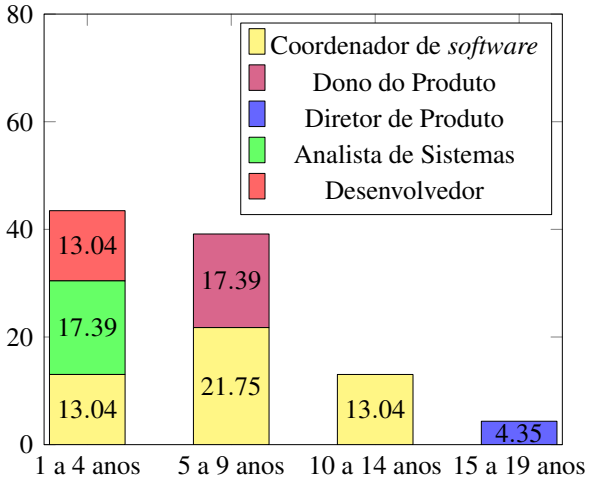
Fonte: Elaborado pelo autor (2023).

Na Figura 28, há a relação entre o cargo atual e o tempo de experiência, sendo que a maioria dos respondentes está no cargo de Coordenador de *software* (47,83%), com menos de nove anos de experiência (82.61%).

Na avaliação dos perfis de respondentes, se conclui que aproximadamente 70% se encontram em cargos de liderança (Diretor de Produto, Coordenador de *software* e Dono do Produto), sendo que, nestes cargos, 56,52% possuem mais de cinco anos na gestão de manutenção de *software*, apresentando maturidade no conhecimento da área.

O Quadro 22 apresenta, em números absolutos, as respostas de todos os participantes, avaliando a escala Likert de 5 (cinco) pontos, sendo que, para cada pontuação, foi atribuído um valor de 1 (um) até 5 (cinco). Já a Figura 29 apresenta os resultados para cada uma das questões do questionário TAM.

Figura 28 – Resultado TAM — Tempo de Experiência por Cargo



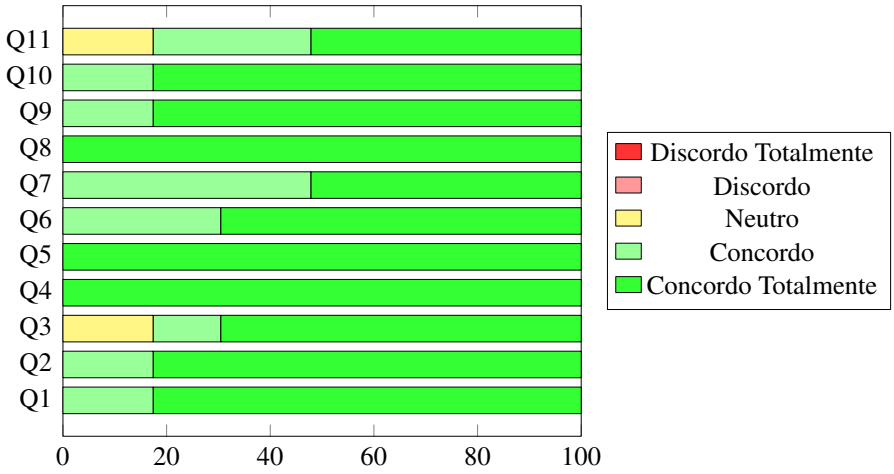
Fonte: Elaborado pelo autor (2023).

Quadro 22 – Resultado TAM — Respostas Questionário TAM para o SMAE

Questão	**Discordo Totalmente**	**Discordo**	**Neutro**	**Concordo**	**Concordo Totalmente**
Q1	0	0	0	4	19
Q2	0	0	0	4	19
Q3	0	0	4	3	16
Q4	0	0	0	0	23
Q5	0	0	0	0	23
Q6	0	0	0	7	16
Q7	0	0	0	11	12
Q8	0	0	0	0	23
Q9	0	0	0	4	19
Q10	0	0	0	4	19
Q11	0	0	4	7	12
Total	0	0	8	44	201

Fonte: Elaborado pelo autor (2023).

Figura 29 – Resultado TAM — Respostas Questionário TAM para o SMAE

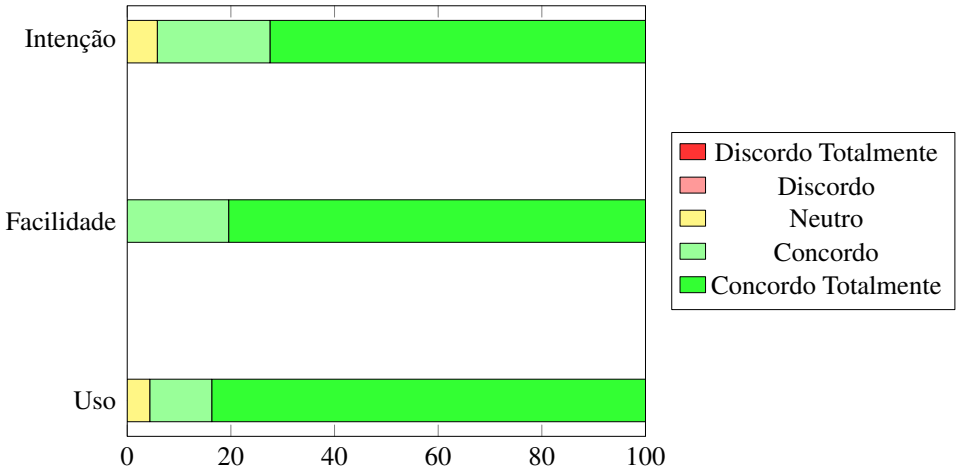


Fonte: Elaborado pelo autor (2023).

Foi observado que em nenhuma das respostas houve discordância dos participantes, sendo que apenas nas questões Q3 e Q11 houve neutralidade, sendo representado por apenas 3.16%. A maioria das respostas apresenta concordância com as percepções de uso, facilidade e intenção, sendo que 79,45% das respostas indicam concordância total (concordo totalmente).

Na Figura 30, é apresentada uma visão sumarizada pelos constructos. O nível de concordância foi considerado positivo, sendo que a distribuição dos percentuais de concordância foi Facilidade com 100%, Uso com 95,65% e Intenção com 94,2%.

Figura 30 – Resultado TAM — Distribuição pelo Constructo



Fonte: Elaborado pelo autor (2023).

Na questão aberta, cujo objetivo era coletar impressões, sugestões de melhoria, houve apenas duas respostas, sendo considerados dois elogios:

- Muito boa a ideia.
- Achei excelente a aplicação, vou sugerir em minha empresa para tentarmos aplicar.

Portanto, pode-se concluir que, com os percentuais claramente favoráveis ao *framework*, este pode ser aplicado como suporte à tomada de decisão dos gestores na área de manutenção de *software*.

5.4 AMEAÇAS À VALIDADE

Com o objetivo de garantir a qualidade da pesquisa realizada, a identificação das ameaças à validade da pesquisa permite a definição de controles e ações para mitigá-las, orientando, assim, a reprodutibilidade da pesquisa.

5.4.1 Tamanho da Amostra

Devido ao tamanho limitado da amostra no experimento, composta por apenas cinco participantes, os resultados podem não ser generalizáveis para uma amostra maior, colocando em risco a validade interna do estudo.

Para mitigar essa ameaça, além da realização do experimento prático, aplicou-se o questionário TAM, visando avaliar a aceitação do *framework* por um grupo distinto daquele no qual o experimento foi realizado. Isso possibilitou um número maior de gestores e permitiu uma aplicação mais ampla do *framework* para uma população maior.

5.4.2 Seleção dos participantes

A ameaça de viés de seleção dos participantes pode ser preocupante em experimentos com um número limitado de respondentes, uma vez que a seleção dos indivíduos pode resultar uma amostra não representativa.

Para mitigar essa ameaça, os gestores selecionados atuam em diferentes empresas, evitando, assim, qualquer viés relacionado aos processos de manutenção de *software* ou aos produtos de manutenção de *software*, garantindo uma abordagem mais equilibrada na escolha dos participantes e reduzindo possíveis distorções nos resultados.

5.4.3 Conscientização da observação

A influência da mudança no comportamento dos participantes, devido à conscientização de estarem sendo observados, pode ser ampliada em experimentos com um número reduzido de participantes. Em grupos tão pequenos, a presença do pesquisador pode se tornar mais evidente, o que pode levar os participantes a modificarem seu comportamento de maneira a não representar a realidade.

Para atenuar esse efeito, as entrevistas foram conduzidas em intervalos temporais menores, permitindo o estabelecimento de uma relação de confiança com os participantes. Além disso, os objetivos do estudo foram explicados claramente, garantindo que a observação não resultaria em consequências negativas para os gestores.

5.4.4 Influência do comportamento do pesquisador

O efeito do experimentador, que acontece quando o comportamento do pesquisador influencia o comportamento dos participantes, é uma ameaça à validade em experimentos com um pequeno número de participantes. A interação direta entre o pesquisador e os participantes pode ser mais proeminente, tornando a possibilidade de influência mais significativa.

Para reduzir esse efeito, os roteiros de entrevista semiestruturada foram padronizados, o que ajudou a minimizar a influência pessoal do pesquisador sobre as respostas dos participantes.

5.4.5 Viés da ferramenta

Dado que todas as avaliações durante a experimentação em ambiente real foram realizadas na ferramenta desenvolvida para a avaliação do *framework*, existe a possibilidade de haver influência da ferramenta desenvolvida. Para mitigar esse viés, durante as entrevistas de acompanhamento, foram enfatizados os pontos de avaliação do *framework*, bem como as questões de avaliação foram direcionadas para os resultados obtidos durante a experimentação e não para a interface de uso.

Como para a avaliação das respostas ao questionário TAM foi utilizado um vídeo demonstrativo da ferramenta, pode-se considerar que o viés visual e narrativo presente no vídeo pode influenciar a percepção dos usuários, introduzindo elementos subjetivos que podem distorcer as avaliações. Para mitigar esses viéses, a estrutura de questões foi organizada para uma coleta de dados objetivos e quantitativos do *framework*, evitando, com isso, respostas direcionadas para a ferramenta.

5.5 CONSIDERAÇÕES DO CAPÍTULO

De maneira a experimentar o *framework* desenvolvido, foi aplicada a ferramenta em um ambiente real, tendo como grupo de cinco equipes distintas da área de manutenção de *software*. Desta forma, os gestores dessa área foram capacitados a utilizar a ferramenta, visando a aplicação durante o planejamento de suas atividades.

Inicialmente, foi realizada uma entrevista semiestruturada para identificar o perfil e a demografia dos gestores, sendo identificado como região de aplicação Joinville/SC, tendo como idade média dos gestores 28 anos, bem como tempo médio de gestão da área de *software* como cinco anos. Foi identificado, ainda, que as empresas possuem atividades estratégicas na área de manutenção de *software*, sendo que o custo aplicável em atividades desta área é elevado.

Ao longo da aplicação do *framework*, todos os gestores perceberam um suporte adicional em suas decisões, reconhecendo que as sugestões fornecidas pelo *framework* contribuíram para aprimorar a dinâmica da área de manutenção de software, resultando maior facilidade na priorização de tarefas. Logo, todos os gestores concordaram com a aplicabilidade do *framework*.

Além disso, foi aplicado o questionário TAM para medir a aceitação do *framework* como um modelo válido para apoio na tomada de decisão de priorização de tarefas e alocação de

recursos. A aplicação do questionário se deu por meio de formulário *online* direcionado ao público-alvo com gestores da área de manutenção de *software*.

O questionário alcançou 23 participantes, sendo que a faixa etária predominante foi entre 25 e 34 anos (60,9%) que possuam graduação (60,87%), indicando um público de média escolaridade. Também, esse público apresentou 70% das respostas em cargos de liderança (Diretor de Produto, Coordenador de *software* e Dono do Produto) e com mais de cinco anos na gestão da área de manutenção de *software*.

Os resultados também apresentaram um nível de concordância médio de 79,45% considerando apenas as respostas do tipo "concordo totalmente". Já as médias de concordância por constructo foram de Facilidade com 100%, Uso com 95,65% e Intenção com 94,2%. Por fim, os resultados dos comentários não trouxeram informações relevantes, sendo considerados apenas elogios.

Diante do exposto, conclui-se que o *framework* foi avaliado por meio da ferramenta desenvolvida e possui aceitação pela área de manutenção de *software* como uma solução viável para a priorização de tarefas e apresentação de sugestões na alocação de recurso, apoiando os gestores de manutenção na tomada de decisão.

6 CONCLUSÕES

A manutenção de *software*, por fazer parte do ciclo de desenvolvimento de *software*, tem caráter relevante na manutenção da qualidade e utilidade do *software*. Logo, garantir uma correta execução nas tarefas de manutenção de *software*, com qualidade e agilidade, é de fundamental importância. Esta pesquisa teve como objetivo a elaboração de um *framework* para auxiliar na tomada de decisão dos gestores da área de manutenção de *software*, a fim de permitir maior assertividade em seus planejamentos e nas entregas da equipe de desenvolvimento.

Com a realização da revisão sistemática da literatura foi possível identificar o cenário atual das métricas ligadas à área de manutenção de *software*, percebendo-se que, no que tange às perspectivas de processos e produtos, as métricas ainda são pouco exploradas (MACHADO; KEMCZINSKI; SCHROEDER, 2023), isso também foi corroborado com a pesquisa exploratória realizada com a indústria, cujos resultados permitem compreender que a indústria também pouco aproveita essas perspectivas. Ainda, outro ponto pouco explorado está na dinamicidade da área de manutenção, a qual possui uma série de atividades a serem realizadas, trazendo maior complexidade para a sua gestão.

Diante disso, foi implementado um *framework* que auxilia na tomada de decisão e permite ao gestor da área de manutenção ter uma visão global de suas atividades, denominado *Software Maintenance Activity Evaluation* - SMAE. O *framework* tem como principal prerrogativa a aplicação equitativa das métricas de produto, de processos e de produto, bem como considera as diferentes etapas aplicadas a cada atividade de manutenção de *software*, permitindo ao gestor avaliar a melhor composição de equipe para determinada tarefa, avaliando a criticidade de entrega de cada uma delas.

Para a avaliação do *framework* SMAE, foi elaborada uma ferramenta *web* com o intuito de demonstrar suas funcionalidades e seus conceitos, possuindo tanto uma interface para utilização, quanto o acionamento de suas estruturas por meio de API (*Application Programming Interface* - Interface de Programação de Aplicativos).

De maneira a verificar a aceitação do *framework*, foi realizada uma aplicação da ferramenta em ambiente controlado, tendo como grupo experimental cinco gestores da área de manutenção de *software* com equipes distintas. A condução da utilização foi acompanhada em todo o processo pelo pesquisador, para garantir a utilização da ferramenta e auxiliar em caso de dúvidas sobre a utilização do sistema. Como conclusão da aplicação, os gestores se mostraram adeptos a utilização do *framework* para auxílio na tomada de decisão, tanto na priorização de tarefas quanto na alocação de recurso, desde que sua aplicação seja feita complementarmente à dos gestores.

Também foi aplicado um questionário TAM para medir a aceitação do *framework*, com 23 participantes da área de manutenção de *software*. Destes, cerca de 70% ocupavam cargos de liderança, bem como possuíam mais de cinco anos na gestão da área de manutenção de *software*, denotando elevado conhecimento na área por parte dos respondentes. O resultado do questionário

TAM apresentou um nível de concordância médio elevado (79,45%), bem como a concordância por constructo foi de Facilidade com 100%, Uso com 95,65% e Intenção com 94,2%.

A problemática, destacada na Seção 1.1, direciona à necessidade de realizar avaliações adequadas na área de manutenção de *software*, em especial, com apoio nas perspectivas de Produto, de Processo e de Recurso, considerando o dinamismo da área.

Com os resultados qualitativos, da aplicação em ambiente controlado, e quantitativos, do questionário TAM, ficou evidente a aceitação do *framework* como ferramenta para apoiar a decisão dos gestores da área de manutenção de *software*, tanto em priorização de tarefas quanto na alocação de recursos. Logo, o *framework* busca resolver a problemática, haja vista que, fornece a visão das três perspectivas em conjunto, fornecendo suporte para a decisão dos gestores.

Além dos resultados acima descritos, o mapeamento sistemático da literatura foi publicado no Simpósio Brasileiro de Sistemas de Informação de 2023 (SBSI 2023), sob o título de *Survey of Software Maintenance Metrics: A Systematic Literature Review*.

Por fim, esta pesquisa teve como motivação uma necessidade do próprio pesquisador, o que gerou resultados positivos e permitiu o aprendizado de diversas áreas de conhecimento, abrindo novos caminhos para estudos e realização de trabalhos futuros com foco no planejamento da área de manutenção de *software*.

6.1 TRABALHOS FUTUROS

Entre os trabalhos futuros, podem ser destacados os pontos de melhoria levantados pelos gestores durante a utilização em ambiente controlado do *framework*, que visa ampliar sua aderência ao ambiente empresarial e facilitar a sua entrada como uma ferramenta de gestão e apoio à decisão. Também, durante o desenvolvimento do *framework*, foram identificadas melhorias que podem ser aproveitadas como avanços nas pesquisas do *framework* SMAE. Desse modo, as possíveis melhorias são:

1. **Processos em outros paradigmas de programação**: foi identificado durante o desenvolvimento que o *framework* é focado no paradigma da programação orientada a objetos, dessa forma, para ampliar a sua utilização, pesquisas sobre os processos de desenvolvimento em outros paradigmas podem se aplicadas para levantar as diferentes métricas e como podem ser aplicadas no cálculo de priorização de tarefa e na sugestão de alocação de recursos.

2. **Integração com plataformas de gestão de código-fonte**: foi identificado durante o desenvolvimento que a implantação do *framework* é dispendiosa, pois as informações devem ser cadastradas manualmente. Logo, a integração com plataformas de gestão de código-fonte, ou até a elaboração de importadores podem facilitar a sua aplicação.

3. **Extração de dados de ferramentas automatizadas**: foi identificado durante o desenvolvimento que a integração com plataformas de avaliação de código-fonte, de processos

ou de recursos, podem melhorar a utilização do *framework*, garantindo a precisão das métricas fornecidas. Desse modo, os gestores não necessitariam atualizar manualmente as informações para atualização dos resultados do *framework*.

4. **Integração com plataformas de gestão de projetos**: foi identificado durante a utilização pelos gestores que a integração com plataformas de gestão, tais como Azure, JIRA e afins, permitem uma melhor gestão da área de manutenção de *software*. Logo, a integração das sugestões do *framework* com essas ferramentas pode criar um ambiente de gestão integrado.

5. **Considerar outros tipos de produto**: foi identificado durante a utilização pelos gestores que os produtos da área de manutenção de *software* podem não necessariamente ser códigos-fonte, podendo ser documentações técnicas, levantamento de requisitos, entre outros. Logo, pesquisas aprofundadas sobre os produtos da área de manutenção de *software* podem trazer relevantes aspectos para a melhoria do *framework*.

6. **Considerar restrições de recurso**: foi identificado durante a utilização pelos gestores a necessidade de restrições de recursos, que podem ocorrer por indisponibilidade do recurso por absenteísmo, férias ou recessos. Desta forma, uma possível melhoria é a criação de períodos de restrição, em que o recurso estaria indisponível e as sugestões de priorização e alocação se comportariam para se adequar a tais restrições.

7. **Sugestão de segundas alocações**: foi identificado durante a utilização pelos gestores a necessidade de apresentar mais de uma sugestão de alocação. Desta forma, o gestor poderá ter mais liberdade na definição de alocação, tendo visões distintas para aplicação.

8. **Integração com redes sociais**: outro ponto que pode trazer melhorias ao *framework* está na aplicação de integração de redes sociais como o LinkedIn para a sugestão de alocação por competência nas etapas ou atividades nas quais os gestores possuem maior déficit de profissionais.

9. **Aplicação de Inteligência Artificial**: os algoritmos de cálculo do *framework* podem ser otimizados com a aplicação de redes neurais ou aprendizado semi-supervisionado. Desse modo, as sugestões apontadas pelo *framework* poderiam ser categorizadas ou aprovadas, de maneira a melhorar a assertividade e reduzir a sua aplicação de forma assistida.

REFERÊNCIAS

AGUIAR, Gilberto de; KEMCZINSKI, Avanilde; GASPARINI, Isabela. **Desenvolvimento de base de dados de atitudes individuais para a formação automatizada de equipes de projetos de software**. Dissertação (Mestrado) — Universidade do Estado de Santa Catarina, 2017. Citado 3 vezes nas páginas 47, 98 e 99.

AHN, Yunsik et al. The software maintenance project effort estimation model based on function points. **Journal of Software Maintenance**, John Wiley & Sons, Inc., USA, v. 15, n. 2, p. 71–85, mar 2003. ISSN 1040-550X. Disponível em: <https://doi.org/10.1002/smr.269>. Citado 2 vezes nas páginas 56 e 129.

AL-JAMIMI, Hamdi A.; AHMED, Moataz. Prediction of software maintainability using fuzzy logic. In: **2012 IEEE International Conference on Computer Science and Automation Engineering**. Beijing, China: IEEE, 2012. p. 702–705. Citado 2 vezes nas páginas 56 e 129.

ALAM, Intakhab; SARWAR, Nadeem; NOREEN, Iram. Statistical analysis of software development models by six-pointed star framework. **PLOS ONE**, Public Library of Science (PLoS), v. 17, n. 4, p. e0264420, abr. 2022. Disponível em: <https://doi.org/10.1371/journal.pone.0264420>. Citado na página 25.

APRIL, Alain; ABRAN, Alain. **Software Maintenance Management : Evaluation and Continuous Improvement**. New Jersey: Wiley-IEEE Computer Society PR, 2008. ISBN 0470147075. Citado 2 vezes nas páginas 42 e 71.

BABBIE, Earl R. **Survey Research Methods**. Wadsworth Publishing Company, 1990. ISBN 9780534126728. Disponível em: <https://books.google.com.br/books?id=EhtHAAAAMAAJ>. Citado na página 59.

BAKOTA, Tibor et al. A probabilistic software quality model. In: . Williamsburg, VA, USA: IEEE, 2011. p. 243–252. Citado 2 vezes nas páginas 56 e 129.

BASEER, K K. A systematic survey on waterfall vs. agile vs. lean process paradigms. **A Systematic Survey on Waterfall vs. Agile vs. Lean Process Paradigms**, v. 9, p. 33–58, 01 2015. Citado 3 vezes nas páginas 25, 26 e 27.

BHATT, Pankaj et al. An influence model for factors in outsourced software maintenance. **Journal of Software Maintenance**, v. 18, p. 385–423, 11 2006. Citado 2 vezes nas páginas 56 e 129.

BIGONHA, Mariza A.S. et al. The usefulness of software metric thresholds for detection of bad smells and fault prediction. **Information and Software Technology**, v. 115, p. 79–92, 2019. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584919301697>. Citado na página 37.

BOEHM, Barry. A view of 20th and 21st century software engineering. In: **Proceedings of the 28th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2006. (ICSE '06), p. 12–29. ISBN 1595933751. Disponível em: <https://doi.org/10.1145/1134285.1134288>. Citado 3 vezes nas páginas 25, 27 e 41.

BOURQUE, Pierre; FAIRLEY, Richard E. (Ed.). **SWEBOK: Guide to the Software Engineering Body of Knowledge**. Version 3.0. Los Alamitos, CA: IEEE Computer Society,

2014. ISBN 978-0-7695-5166-1. Disponível em: <http://www.swebok.org/>. Citado 5 vezes nas páginas 29, 33, 34, 41 e 42.

BUCHINGER, Diego; CAVALCANTI, Gustavo; HOUNSELL, Marcelo. Mecanismos de busca acadêmica: uma análise quantitativa. **Revista Brasileira de Computação Aplicada**, v. 6, n. 1, p. 108–120, abr. 2014. Disponível em: <http://seer.upf.br/index.php/rbca/article/view/3452>. Citado na página 50.

BURROWS, Rachel et al. An empirical evaluation of coupling metrics on aspect-oriented programs. In: **Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics**. New York, NY, USA: Association for Computing Machinery, 2010. (WETSoM '10), p. 53–58. ISBN 9781605589763. Disponível em: <https://doi.org/10.1145/1809223.1809231>. Citado 2 vezes nas páginas 56 e 129.

BöRSTLER, Jürgen; CASPERSEN, Michael; NORDSTRöM, Marie. Beauty and the beast: on the readability of object-oriented example programs. **Software Quality Journal**, v. 24, 02 2015. Citado 2 vezes nas páginas 56 e 129.

CALERO, Coral; BERTOA, Manuel; MORAGA, Maria. A systematic literature review for software sustainability measures. **2013 2nd International Workshop on Green and Sustainable Software, GREENS 2013 - Proceedings**, p. 46–53, 05 2013. Citado na página 18.

CANFORA, G. et al. A family of experiments to validate metrics for software process models. **J. Syst. Softw.**, Elsevier Science Inc., USA, v. 77, n. 2, p. 113–129, aug 2005. ISSN 0164-1212. Disponível em: <https://doi.org/10.1016/j.jss.2004.11.007>. Citado 2 vezes nas páginas 57 e 129.

CHOUDHARI, Jitender; SUMAN, Ugrasen. Story points based effort estimation model for software maintenance. **Procedia Technology**, v. 4, p. 761–765, 12 2012. Citado 2 vezes nas páginas 57 e 129.

CHUG, Anuradha; MALHOTRA, R. Benchmarking framework for maintainability prediction of open source software using object oriented metrics. **International Journal of Innovative Computing, Information and Control - ICIC International 2016**, v. 12, p. 615–634, 01 2016. ISSN 1349-4198. Citado na página 70.

CIOLKOWSKI, Marcus et al. Practical experiences in the design and conduct of surveys in empirical software engineering. In: **Empirical Methods and Studies in Software Engineering**. Springer Berlin Heidelberg, 2003. p. 104–128. Disponível em: <https://doi.org/10.1007/978-3-540-45143-3_7>. Citado 3 vezes nas páginas 21, 59 e 60.

CONCAS, Giulio et al. Assessing traditional and new metrics for object-oriented systems. In: **Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics**. New York, NY, USA: Association for Computing Machinery, 2010. (WETSoM '10), p. 24–31. ISBN 9781605589763. Disponível em: <https://doi.org/10.1145/1809223.1809227>. Citado 2 vezes nas páginas 57 e 129.

COUTO, Christian et al. Mcl: Metrics-based constraint language. In: . New York, NY, USA: Association for Computing Machinery, 2018. p. 1–8. ISBN 978-1-4503-6559-8. Citado 2 vezes nas páginas 57 e 129.

DAGPINAR, M.; WEBER, Jens. Predicting maintainability with object-oriented metrics - an empirical comparison. In: . Victoria, BC, Canada: IEEE, 2003. p. 155– 164. ISBN 0-7695-2027-8. Citado 2 vezes nas páginas 57 e 129.

DAHAB, Sarah; PORRAS, Juan; MAAG, Stephane. A novel formal approach to automatically suggest metrics in software measurement plans. In: . Madeira, Portugal: SciTePress, 2018. p. 283–290. Citado 2 vezes nas páginas 57 e 129.

DAHAB, Sarah et al. Enhancing software development process quality based on metrics correlation and suggestion. In: . Porto, Portugal: SciTePress, 2018. p. 154–165. Citado na página 38.

DANTAS, Francisco; GARCIA, Alessandro; WHITTLE, Jon. On the role of composition code properties on evolving programs. In: **Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: Association for Computing Machinery, 2012. (ESEM '12), p. 291–300. ISBN 9781450310567. Disponível em: <https://doi.org/10.1145/2372251.2372304>. Citado 2 vezes nas páginas 57 e 129.

DAVIS, Fred D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS Q.**, v. 13, p. 319–340, 1989. Citado 3 vezes nas páginas 47, 98 e 99.

DEMING, W.E. **Out of the Crisis**. Cambridge, MA, USA: Massachusetts Institute of Technology, Center for Advanced Engineering Study, 2000. (Mit Press). ISBN 9780262541152. Disponível em: <https://books.google.com.br/books?id=LA15eDlOPgoC>. Citado na página 37.

DIMA, Alina Mihaela; MAASSEN, Maria Alexandra. From waterfall to agile software: Development models in the IT sector, 2006 to 2018. impacts on company management. **Journal of International Studies**, Centre of Sociological Research, NGO, v. 11, n. 2, p. 315–326, jun. 2018. Disponível em: <https://doi.org/10.14254/2071-8330.2018/11-2/21>. Citado na página 26.

DRESCH, Aline; LACERDA, Daniel; ANTUNES, Junico. **Design Science Research: Método de Pesquisa para Avanço da Ciência e Tecnologia**. Porto Alegre: Bookman, 2015. ISBN 978-85-8260-298-0. Citado na página 44.

EJIOGU, Lem O. Five principles for the formal validation of models of software metrics. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 28, n. 8, p. 67–76, ago. 1993. ISSN 0362-1340. Disponível em: <https://doi.org/10.1145/163114.163123>. Citado 2 vezes nas páginas 51 e 52.

EL-SHARKAWY, Sascha; KRAFCZYK, Adam; SCHMID, Klaus. Fast static analyses of software product lines: An example with more than 42,000 metrics. In: **Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems**. New York, NY, USA: Association for Computing Machinery, 2020. (VAMOS '20). ISBN 9781450375016. Disponível em: <https://doi.org/10.1145/3377024.3377031>. Citado 2 vezes nas páginas 57 e 129.

FENTON, Norman E.; PFLEEGER, Shari Lawrence. **Software Metrics: A Rigorous and Practical Approach (2nd ed. Revisited Printing)**. London: International Thomson Computer Press., 2004. Citado 2 vezes nas páginas 17 e 38.

FILHO, Wilson De Pádua Paula. **Engenharia de Software: Fundamentos, Métodos e Padrões**. Rio de Janeiro: LTC, 2001. Citado na página 24.

FIORAVANTI, F.; NESI, P.; STORTONI, F. Metrics for controlling effort during adaptive maintenance of object oriented systems. In: **Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)**. Oxford, UK: IEEE, 1999. p. 483–492. Citado 2 vezes nas páginas 57 e 129.

FLORIANI, Eduarda Vieira; STEIL, Andrea Valeria. Processos de aprendizagem em equipe de projeto que utiliza metodologia agil/learning processes in a project team using agile methodology. **Revista de Gestão e Projetos**, Universidade Nove de Julho, v. 12, n. 1, p. 149, 2021. ISSN 2236-0972. Citado na página 28.

FRANTZ, Rafael Z. et al. Ranking open source application integration frameworks based on maintainability metrics: A review of five-year evolution. **Software: Practice and Experience**, v. 49, n. 10, p. 1531–1549, 2019. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2733>. Citado 2 vezes nas páginas 41 e 69.

GARCÍA, Félix; RUIZ, Francisco; PIATTINI, Mario. Definition and empirical validation of metrics for software process models. In: BOMARIUS, Frank; IIDA, Hajimu (Ed.). **Product Focused Software Process Improvement**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 146–158. ISBN 978-3-540-24659-6. Citado 2 vezes nas páginas 57 e 129.

GARCíA-BORGOñóN, L. et al. Software process modeling languages: A systematic literature review. **Information and Software Technology**, v. 56, n. 2, p. 103–116, 2014. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584913001894>. Citado na página 23.

GARCíA, Félix et al. Towards a consistent terminology for software measurement. **Information and Software Technology**, v. 48, n. 8, p. 631 – 644, 2006. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584905001011>. Citado na página 40.

GEZICI, Bahar; TARHAN, Ayça; CHOUSEINOGLOU, Oumout. Internal and external quality in the evolution of mobile software: An exploratory study in open-source market. **Information and Software Technology**, v. 112, p. 178–200, 2019. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584918301290>. Citado na página 32.

GHANEM, Timothy. Predicting software maintenance activities. 02 2020. Citado 2 vezes nas páginas 43 e 86.

GIL, A.C. **Métodos e ténicas de pesquisa social**. São Paulo: Atlas, 2008. ISBN 9788522431694. Citado na página 20.

GIL, A.C. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 2017. ISBN 9788522431694. Citado 2 vezes nas páginas 20 e 58.

GIL, Joseph; GOLDSTEIN, Maayan; MOSHKOVICH, Dany. An empirical investigation of changes in some software properties over time. In: **2012 9th IEEE Working Conference on Mining Software Repositories (MSR)**. Zurich, Switzerland: IEEE, 2012. p. 227–236. Citado 2 vezes nas páginas 57 e 129.

GIL, Yossi; GOLDSTEIN, Maayan; MOSHKOVICH, Dany. How much information do software metrics contain? In: **Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools**. New York, NY, USA: Association for Computing Machinery, 2011. (PLATEAU '11), p. 57–64. ISBN 9781450310246. Disponível em: <https://doi.org/10.1145/2089155.2089169>. Citado 2 vezes nas páginas 57 e 129.

GORDEA, Sergiu; ZANKER, Markus. Building maintenance charts and early warning about scheduling problems in software projects. In: . Setúbal, Portugal: SciTePress, 2006. v. 1, p. 210–217. Citado 2 vezes nas páginas 57 e 129.

HAYES, Jane; MOHAMED, Naresh; GAO, Tina. Observe-mine-adopt (oma): An agile way to enhance software maintainability. **Journal of Software Maintenance**, v. 15, p. 297–323, 09 2003. Citado 2 vezes nas páginas 57 e 129.

HAYES, J.H.; PATEL, S.C.; ZHAO, L. A metrics-based software maintenance effort model. In: **Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.** Tampere, Finland: IEEE, 2004. p. 254–258. Citado 2 vezes nas páginas 57 e 129.

HEGEDüS, Péter et al. A drill-down approach for measuring maintainability at source code element level. In: . Berlin, Germany: EASST, 2013. Citado 2 vezes nas páginas 57 e 129.

HEVNER, Alan et al. Design science in information systems research. **Management Information Systems Quarterly**, v. 28, p. 75–, 03 2004. Citado 3 vezes nas páginas 44, 45 e 92.

HIRAMA, Kenchi. **Engenharia de software: qualidade e produtividade com tecnologia**. Rio de Janeiro: LTC, 2011. ISBN 97885352488821. Citado na página 16.

HOVSEPYAN, Aram et al. From aspect-oriented models to aspect-oriented code?: The maintenance perspective. In: . New York, NY, USA: Association for Computing Machinery, 2010. p. 85–96. Citado 2 vezes nas páginas 57 e 129.

HUANG, Sun-Jen; LAI, R. Measuring the maintainability of a communication protocol based on its formal specification. **IEEE Transactions on Software Engineering**, v. 29, n. 4, p. 327–344, 2003. Citado 2 vezes nas páginas 58 e 130.

HUANG, Xin et al. Synthesizing qualitative research in software engineering: A critical review. Association for Computing Machinery, New York, NY, USA, p. 1207–1218, 2018. Disponível em: <https://doi.org/10.1145/3180155.3180235>. Citado na página 50.

HUDA, Shamsul et al. A framework for software defect prediction and metric selection. **IEEE Access**, v. 6, p. 2844–2858, 2018. Citado 3 vezes nas páginas 38, 40 e 41.

HUNT, Andrew; THOMAS, David. **O programador pragmático: de aprendiz a mestre**. Porto Alegre: Bookman, 2010. ISBN 9788577807345. Citado na página 18.

IQBAL, Nayyar et al. Measuring software maintainability with naïve bayes classifier. **Entropy**, v. 23, n. 2, 2021. ISSN 1099-4300. Disponível em: <https://www.mdpi.com/1099-4300/23/2/136>. Citado 3 vezes nas páginas 36, 42 e 71.

ISO/IEC. **15939:2007, Software Engineering-Software Measurement Process**. Geneva, Switzerland, 2007. Citado 2 vezes nas páginas 40 e 54.

JAIN, Ashu; TARWANI, Sandhya; CHUG, Anuradha. An empirical investigation of evolutionary algorithm for software maintainability prediction. In: **2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS)**. Bhopal, India: IEEE, 2016. p. 1–6. Citado 4 vezes nas páginas 16, 18, 57 e 129.

JONES, T. Capers. Estimating software costs. 1998. Citado 3 vezes nas páginas 43, 71 e 85.

JøRGENSEN, Magne; SJøBERG, Dag I. K. Impact of experience on maintenance skills. **Journal of Software Maintenance and Evolution: Research and Practice**, v. 14, n. 2, p. 123–146, 2002. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.248>. Citado 2 vezes nas páginas 57 e 129.

KHEZAMI, Nadhira; KESSENTINI, Marouane; FERREIRA, Thiago Do N. A systematic literature review on software maintenance for cyber-physical systems. **IEEE Access**, v. 9, p. 159858–159872, 2021. Citado na página 29.

KITCHENHAM, Barbara. **Measuring Software Development**. USA: Elsevier Science Inc., 1990. ISBN 1851664009. Citado na página 38.

KITCHENHAM, Barbara;; CHARTERS, Stuart. Guidelines for performing systematic literature reviews in software engineering. v. 2, 01 2007. Citado na página 50.

KITCHENHAM, Barbara; PICKARD, Lesley; PFLEEGER, Shari Lawrence. Case studies for method and tool evaluation. **IEEE Software**, IEEE, London, UK, v. 12, n. 4, p. 52–62, 1995. Citado 2 vezes nas páginas 58 e 59.

KUHRMANN, Marco et al. Flexible software process lines in practice: A metamodel-based approach to effectively construct and manage families of software process models. **Journal of Systems and Software**, v. 121, p. 49–71, 2016. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121216301236>. Citado na página 24.

KULKARNI, U.L.; KALSHETTY, Y.R.; ARDE, Vrushali G. Validation of ck metrics for object oriented design measurement. In: **2010 3rd International Conference on Emerging Trends in Engineering and Technology**. Goa, India: IEEE, 2010. p. 646–651. Citado 2 vezes nas páginas 57 e 129.

LEHMAN, M. M.; BELADY, L. A. **Program Evolution: Processes of Software Change**. USA: Academic Press Professional, Inc., 1985. ISBN 0124424406. Citado 3 vezes nas páginas 16, 31 e 32.

LEVIN, Stanislav; YEHUDAI, Amiram. Visually exploring software maintenance activities. **CoRR**, abs/1910.08907, 2019. Disponível em: <http://arxiv.org/abs/1910.08907>. Citado 3 vezes nas páginas 33, 34 e 42.

LEWIS, J. A.; HENRY, S. M. On the benefits and difficulties of a maintainability via metrics methodology. **Journal of Software Maintenance**, John Wiley & Sons, Inc., USA, v. 2, n. 2, p. 113–142, jun 1990. ISSN 1040-550X. Disponível em: <https://doi.org/10.1002/smr.4360020203>. Citado 2 vezes nas páginas 57 e 129.

LI, Wei; HENRY, Sallie. Object-oriented metrics that predict maintainability. **Journal of Systems and Software**, v. 23, n. 2, p. 111–122, 1993. ISSN 0164-1212. Object-Oriented Software. Disponível em: <https://www.sciencedirect.com/science/article/pii/016412129390077B>. Citado 2 vezes nas páginas 57 e 129.

LILJA, D.J. **Measuring Computer Performance: A Practitioner's Guide**. Cambridge University Press, 2005. ISBN 9780521646703. Disponível em: <https://books.google.com.br/books?id=R8RLniX5DNQC>. Citado na página 72.

MACHADO, Jackson; KEMCZINSKI, Avanilde; SCHROEDER, Rebeca. Survey of software maintenance metrics: A systematic literature review. In: **Proceedings of the XIX Brazilian Symposium on Information Systems**. New York, NY, USA: Association for Computing Machinery, 2023. (SBSI '23), p. 332–339. ISBN 9798400707599. Disponível em: <https://doi.org/10.1145/3592813.3592922>. Citado 9 vezes nas páginas 49, 50, 51, 52, 53, 54, 55, 56 e 111.

MAL, Sandip; RAJNISH, Kumar. New class cohesion metric: An empirical view. **International Journal of Multimedia and Ubiquitous Engineering**, v. 9, p. 367–376, 06 2014. Citado 2 vezes nas páginas 57 e 129.

MALHOTRA, Ruchika; CHUG, Anuradha. Comparative analysis of agile methods and iterative enhancement model in assessment of software maintenance. In: **2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)**. New Delhi, India: IEEE, 2016. p. 1271–1276. Citado na página 28.

MAMUN, Md Abdullah Al; BERGER, Christian; HANSSON, Jörgen. Correlations of software code metrics: An empirical study. In: **Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement**. New York, NY, USA: Association for Computing Machinery, 2017. (IWSM Mensura '17), p. 255–266. ISBN 9781450348539. Disponível em: <https://doi.org/10.1145/3143434.3143445>. Citado 2 vezes nas páginas 57 e 129.

MASSO, Jhon et al. Risk management in the software life cycle: A systematic literature review. **Computer Standards & Interfaces**, v. 71, p. 103431, 2020. ISSN 0920-5489. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0920548919300881>. Citado na página 25.

MEIDAN, Ayman et al. Measuring software process: A systematic mapping study. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 3, jun. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3186888>. Citado 5 vezes nas páginas 17, 23, 24, 70 e 90.

MOLNAR, Arthur-Jozsef; MOTOGNA, Simona. Longitudinal evaluation of open-source software maintainability. **CoRR**, abs/2003.00447, 2020. Disponível em: <https://arxiv.org/abs/2003.00447>. Citado na página 36.

MOLNAR, Arthur-Jozsef; NEAMTU, Alexandra; MOTOGNA, Simona. Evaluation of software product quality metrics. **CoRR**, abs/2009.01557, 2020. Disponível em: <https://arxiv.org/abs/2009.01557>. Citado 3 vezes nas páginas 18, 41 e 69.

MURRAY, Jacqueline. Likert data: What to use, parametric or non-parametric? **International Journal of Business and Social Science**, v. 21, 11 2013. Citado na página 66.

NABILAH; SUNINDYO, Wikan Danar. Controlling software evolution process using code smell visualization. In: **Proceedings of the 2nd International Conference on Control and Computer Vision**. New York, NY, USA: Association for Computing Machinery, 2019. (ICCCV 2019), p. 51–54. ISBN 9781450363228. Disponível em: <https://doi-org.ez74.periodicos.capes.gov.br/10.1145/3341016.3341026>. Citado 2 vezes nas páginas 31 e 35.

OSTBERG, Jan-Peter; WAGNER, Stefan. On automatically collectable metrics for software maintainability evaluation. In: **2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement**. Rotterdam, Netherlands: IEEE, 2014. p. 32–37. Citado 3 vezes nas páginas 17, 57 e 129.

PADUELLI, M. M. **Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica**. Dissertação (Mestrado) — USP, SP, 2007. Citado na página 18.

PAZIN, Maicon G.; ALLIAN, Ana P.; JR., Edson Oliveira. Empirical study on software process variability modelling with smartyspem and vspem. **IET Software**, v. 12, n. 6, p. 536–546, 2018. Disponível em: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2017.0061>. Citado na página 25.

PEFFERS, Ken et al. A design science research methodology for information systems research. **Journal of Management Information Systems**, v. 24, p. 45–77, 01 2007. Citado 7 vezes nas páginas 44, 45, 46, 47, 71, 72 e 76.

PERKUSICH, Mirko et al. A procedure to detect problems of processes in software development projects using bayesian networks. **Expert Systems with Applications**, v. 42, n. 1, p. 437–450, 2015. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417414004916>. Citado na página 24.

PETERSEN, Kai et al. Systematic mapping studies in software engineering. **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**, v. 17, 06 2008. Citado 2 vezes nas páginas 21 e 49.

PETERSEN, Kai; WOHLIN, Claes. Context in industrial software engineering research. **2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009**, 10 2009. Citado na página 42.

PFLEEGER, Shari Lawrence. Experimental design and analysis in software engineering. **Annals of Software Engineering**, Springer Science and Business Media LLC, v. 1, n. 1, p. 219–253, dez. 1995. Disponível em: <https://doi.org/10.1007/bf02249052>. Citado na página 58.

PFLEEGER, Shari Lawrence. **Engenharia de software: teoria e prática**. Prentice Hall, 2004. ISBN 9788587918314. Disponível em: <https://books.google.com.br/books?id=MKhmPgAACAAJ>. Citado 7 vezes nas páginas 29, 33, 35, 37, 40, 41 e 42.

PIMENTEL, Mariano; FILIPPO, Denise; SANTORO, Flávia Maria. Design science research: fazendo pesquisas científicas rigorosas atreladas ao desenvolvimento de artefatos computacionais projetados para a educação. In: JAQUES, Patricia et al. (Ed.). **Metodologia de Pesquisa Científica em Informática na Educação: Concepção de Pesquisa**. Porto Alegre: SBC, 2020. v. 1, cap. 5, p. 1–29. ISBN 978-85-7669-493-9. Citado 4 vezes nas páginas 21, 44, 45 e 46.

POLO, Macario et al. Roles in the maintenance process. **ACM SIGSOFT Software Engineering Notes**, v. 24, p. 84–86, 07 1999. Citado na página 29.

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. Porto Alegre: AMGH, 2011. ISBN 978-85-8055-044-3. Citado 12 vezes nas páginas 16, 23, 24, 25, 26, 27, 29, 30, 36, 38, 39 e 41.

RAZANI, Zohreh; KEYVANPOUR, MohammadReza. Sbsr solution evaluation: Methods and challenges classification. In: **2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)**. Tehran, Iran: IEEE, 2019. p. 181–188. Citado 2 vezes nas páginas 57 e 129.

ROCHE, John M. Software metrics and measurement principles. **SIGSOFT Softw. Eng. Notes**, Association for Computing Machinery, New York, NY, USA, v. 19, n. 1, p. 77–85, jan 1994. ISSN 0163-5948. Disponível em: <https://doi.org/10.1145/181610.181625>. Citado 2 vezes nas páginas 38 e 39.

SABOE, M. The use of software quality metrics in the materiel release process experience report. In: **Proceedings Second Asia-Pacific Conference on Quality Software**. Hong Kong, China: IEEE, 2001. p. 104–109. Citado 2 vezes nas páginas 57 e 129.

SAIFAN, Ahmad A.; AL-RABADI, Areej. Evaluating maintainability of android applications. In: **2017 8th International Conference on Information Technology (ICIT)**. Amman, Jordan: IEEE, 2017. p. 518–523. Citado 2 vezes nas páginas 57 e 129.

SANGWAN, Om; SINGH, Pradeep. Aspect oriented software metrics based maintainability assessment: Framework and model. In: . Noida: IET, 2013. v. 2013, p. 1.07–1.07. ISBN 978-1-84919-846-2. Citado 2 vezes nas páginas 57 e 129.

SCHNAPPINGER, Markus et al. Learning a classifier for prediction of maintainability based on static analysis tools. In: **2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)**. Montreal, QC, Canada: IEEE, 2019. p. 243–248. Citado 4 vezes nas páginas 16, 20, 57 e 129.

SELVARANI, R.; NAIR, T.R. Gopalakrishnan; PRASAD, V. Kamakshi. Estimation of defect proneness using design complexity measurements in object-oriented software. In: **2009 International Conference on Signal Processing Systems**. Singapore: IEEE, 2009. p. 766–770. Citado 2 vezes nas páginas 57 e 129.

SHARMA, Arun; BHATIA, Rajesh; GROVER, P. Empirical evaluation and validation of interface complexity metrics for software components. **International Journal of Software Engineering and Knowledge Engineering**, v. 18, p. 919–931, 11 2008. Citado 2 vezes nas páginas 58 e 129.

SHELDON, F.T.; JERATH, Kshamta; CHUNG, Hong. Metrics for maintainability of class inheritance hierarchies. **Journal of Software Maintenance**, v. 14, p. 147–160, 05 2002. Citado 2 vezes nas páginas 58 e 129.

SIAVVAS, Miltiadis G.; CHATZIDIMITRIOU, Kyriakos C.; SYMEONIDIS, Andreas L. Qatch - an adaptive framework for software product quality assessment. **Expert Systems with Applications**, v. 86, p. 350–366, 2017. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417417303883>. Citado na página 36.

SOMMERVILLE, I. **Engenharia de software**. Pearson, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado 17 vezes nas páginas 16, 23, 24, 25, 26, 28, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40 e 41.

Stack Overflow. **Stack Overflow Developer Survey 2021**. 2022. Disponível em: <https://insights.stackoverflow.com/survey/2021>. Citado na página 78.

SUN, Xiaobing et al. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. **Information and Software Technology**, v. 66, p. 1–12, 2015. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584915001007>. Citado na página 29.

SURESH, Yeresime; PATI, Jayadeep; RATH, Santanu. Effectiveness of software metrics for object-oriented system. **Procedia Technology**, v. 6, p. 420–427, 12 2012. Citado 2 vezes nas páginas 58 e 130.

TARWANI, Sandhya; CHUG, Anuradha. Agile methodologies in software maintenance: a systematic review. v. 40, n. 4, p. 415+, Dec 2016. ISSN 03505596. Report. Disponível em: <https://link.gale.com/apps/doc/A600697223/AONE?u=capes&sid=bookmark-AONE&xid=2039efa6>. Citado na página 35.

THONGMAK, Mathupayas; MUENCHAISRI, Pornsiri. Maintainability metrics for aspect-oriented software. **International Journal of Software Engineering and Knowledge Engineering**, v. 19, p. 389–420, 05 2009. Citado 2 vezes nas páginas 58 e 130.

ULZIIT, Bayarbuyan et al. A conceptual framework of challenges and solutions for managing global software maintenance. **Journal of Software: Evolution and Process**, v. 27, n. 10, p. 763–792, 2015. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1720>. Citado na página 29.

VALENTE, Marco Tulio. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**. UmLivro, 2020. ISBN 978-65-00-01950-6. Disponível em: <https://engsoftmoderna.info/>. Citado 2 vezes nas páginas 24 e 28.

VARGA, Ervin. **Unraveling Software Maintenance and Evolution: Thinking Outside the Box**. 1st. ed. New York: Springer Publishing Company, Incorporated, 2018. ISBN 3319713027. Citado 3 vezes nas páginas 30, 42 e 86.

WAZLAWICK, Raul Sidnei. Uma reflexão sobre a pesquisa em ciência da computação à luz da classificação das ciências e do método científico. **Revista de Sistemas de Informação da FSMA**, p. –, 01 2010. Citado na página 20.

WAZLAWICK, Raul Sidnei. **Engenharia de software: conceitos e práticas**. Rio de Janeiro: Elsevier, 2013. ISBN 9788535260847. Citado 15 vezes nas páginas 16, 20, 23, 25, 27, 28, 31, 35, 36, 37, 38, 41, 43, 71 e 85.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. Rio de Janeiro: Elsevier, 2014. Citado na página 93.

WIERINGA, Roel J. **Design Science Methodology for Information Systems and Software Engineering**. Springer Berlin Heidelberg, 2014. Disponível em: <https://doi.org/10.1007/978-3-662-43839-8>. Citado na página 44.

WOHLIN, Claes; HÖST, Martin; HENNINGSSON, Kennet. **Empirical Research Methods in Software Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. 7–23 p. ISBN 978-3-540-45143-3. Disponível em: <https://doi.org/10.1007/978-3-540-45143-3_2>. Citado 2 vezes nas páginas 59 e 60.

WU, Hong et al. Maintenance effort estimation for open source software: A systematic literature review. p. 32–43, 10 2016. Citado na página 70.

ZHANG, Feng et al. How does context affect the distribution of software maintainability metrics? In: **2013 IEEE International Conference on Software Maintenance**. Eindhoven, Netherlands: IEEE, 2013. p. 350–359. Citado 2 vezes nas páginas 58 e 130.

ÇALIKLI, Gül; STARON, Miroslaw; MEDING, Wilhelm. Measure early and decide fast: Transforming quality management and measurement to continuous deployment. In: **Proceedings of the 2018 International Conference on Software and System Process**. New York, NY, USA: Association for Computing Machinery, 2018. (ICSSP '18), p. 51–60. ISBN 9781450364591. Disponível em: <https://doi.org/10.1145/3202710.3203156>. Citado 2 vezes nas páginas 56 e 129.

GLOSSÁRIO

API: A sigla API deriva da expressão inglesa Application Programming Interface que, traduzida para o português, pode ser compreendida como uma interface de programação de aplicação.
ASP: *Adjusted Story Points* — Os Pontos de História Ajustados são uma métrica usada em métodos ágeis de gerenciamento de projetos de software, como o *Scrum*. Os ASP representam a estimativa da complexidade e do esforço necessário para implementar um conjunto de funcionalidades (geralmente chamado de histórias de usuário) após considerar fatores de ajuste, como riscos, incertezas ou complexidades adicionais.
BDD: Complexidade de Domínio do Sistema — A Complexidade de Domínio do Sistema (ou *Business Domain Complexity*) é uma métrica que mede o grau de complexidade do domínio de negócios ao qual o sistema de software está relacionado. Quanto mais complexo o domínio, maior o valor de BDD.
CBO: Acoplamento entre classes — Refere-se à quantidade de classes às quais uma classe está acoplada. Um alto valor de CBO pode indicar uma classe altamente acoplada e, portanto, menos coesa.
CC: Complexidade da Classe — Mede a complexidade de uma classe em relação ao seu tamanho. Uma alta complexidade em relação ao tamanho pode indicar uma classe complexa em relação ao seu volume de código.
Cloud: É um termo aplicável para a disponibilidade sob demanda de recursos do sistema de computador, especialmente armazenamento de dados e capacidade de computação, sem o gerenciamento ativo direto do utilizador.
CONF: Confiança na Resolução de Tarefas — A Confiança na Resolução de Tarefas é uma métrica que avalia o grau de confiança ou certeza de que as tarefas no desenvolvimento ou manutenção de software serão resolvidas com sucesso.
DEVMSYS Experiência de Desenvolvimento de Módulo — Essa métrica mede a experiência de um desenvolvedor em relação ao desenvolvimento de módulos específicos dentro de um sistema de software.
DEVSYS Experiência Total de Desenvolvimento — A Experiência Total de Desenvolvimento representa a experiência acumulada de um desenvolvedor ao trabalhar em projetos de desenvolvimento de software, levando em consideração todos os módulos e aspectos do desenvolvimento.
DIT: Profundidade na Árvore de Herança — Mede a profundidade da árvore de herança de uma classe em uma hierarquia de classes. Quanto mais profunda a árvore, maior o valor de DIT.
DOC: Atualização de Documentação — Essa métrica indica a extensão das atualizações e revisões feitas na documentação relacionada ao software, o que é importante para manter a documentação precisa e útil ao longo do tempo.
D: Dificuldade de Desenvolvimento — A Dificuldade de Desenvolvimento é uma métrica que avalia o grau de complexidade e desafio envolvido no processo de desenvolvimento de software. Ela pode ser usada para estimar a complexidade de tarefas específicas ou do projeto.

ESS Experiência com Software — A Experiência com Software mede o nível de experiência geral de um profissional de software em relação a vários aspectos da engenharia de software, incluindo práticas, tecnologias e ferramentas.
E: Esforço de Trabalho — O Esforço de Trabalho é uma métrica que quantifica o tempo e os recursos necessários para completar um projeto de desenvolvimento ou manutenção de software. Isso inclui o trabalho de programadores, testadores, gerentes de projeto e outros envolvidos no processo.
FPL Familiaridade com a Linguagem de Desenvolvimento — A Familiaridade com a Linguagem de Desenvolvimento avalia o conhecimento e a habilidade de um desenvolvedor com uma linguagem de programação específica. Isso pode influenciar a eficiência e a qualidade do código produzido.
LCOM: Falta de Coesão de Métodos — Essa métrica avalia a coesão de métodos em uma classe, medindo a quantidade de métodos que não compartilham atributos. Um alto valor de LCOM indica uma classe com baixa coesão.
LOC: Linhas de Código — É uma métrica que quantifica o tamanho de um programa ou componente de software contando o número de linhas de código-fonte.
MEXPAPP Experiência de Manutenção na Aplicação — Essa métrica mede a experiência de um profissional em relação à manutenção de uma aplicação de software específica, considerando as tarefas de correção de erros e aprimoramentos.
MEXPTOT Experiência Total de Manutenção — A Experiência Total de Manutenção representa a experiência acumulada de um profissional ao trabalhar em projetos de manutenção de software, considerando todas as aplicações e aspectos da manutenção.
MI: Índice de Manutenibilidade — O Índice de Manutenibilidade é uma métrica que avalia a facilidade com que um software pode ser mantido e modificado. Geralmente, é calculado com base em critérios como complexidade, coesão e acoplamento do código-fonte.
NDA: Número de Dependências de Precedência entre Atividades — O Número de Dependências de Precedência entre Atividades é uma métrica que conta quantas atividades de engenharia de software dependem de outras atividades para serem concluídas antes de começar.
NDWPIn: Número de Dependências de Entrada — Indica quantos produtos de trabalho são necessários como entrada para uma atividade.
NDWPOut: Número de Dependências de Saída — Indica quantos produtos de trabalho são produzidos como saída de uma atividade.
NDWP: Número de Dependências entre Produtos de Trabalho e Atividades — O Número de Dependências entre Produtos de Trabalho e Atividades é uma métrica que avalia as relações de dependência entre os produtos de trabalho (artefatos) e as atividades em um processo de engenharia de software.
NOA: Número de Atributos — Conta a quantidade de atributos ou variáveis de instância em uma classe. Um alto valor de NOA pode indicar uma classe com muitos atributos, o que pode afetar sua coesão e acoplamento.

NOC: Número de Filhos — Indica quantas subclasses herdam de uma classe. Quanto mais subclasses, maior o valor de NOC.
NPR: Número de Funções que Participam do Processo — O Número de Funções que Participam do Processo é uma métrica que indica quantas funções ou partes do sistema estão envolvidas em um determinado processo de engenharia de software. Isso pode ajudar a determinar o escopo e a complexidade das atividades de desenvolvimento ou manutenção.
N: Tamanho do Programa — Refere-se ao tamanho geral do programa, muitas vezes medido em termos de linhas de código total, incluindo comentários e espaços em branco.
RFC: Respostas para a Classe — Conta o número de métodos que podem ser invocados em uma classe, incluindo métodos da própria classe e herdados. Um alto valor de RFC pode indicar uma classe com muitas responsabilidades.
SES: Conformidade com os Padrões de Engenharia de Software — A Conformidade com os Padrões de Engenharia de Software avalia o grau em que o processo ou o produto de software segue os padrões e as melhores práticas estabelecidos na engenharia de software. Isso ajuda a garantir a qualidade e a consistência do trabalho.
Software: É um conjunto de programas, instruções e dados que permitem que um computador execute tarefas específicas. Em termos simples, é o componente intangível de um computador que controla e executa funções
TST: Testabilidade — A Testabilidade é uma métrica que avalia a facilidade com que um sistema de software pode ser testado. Quanto mais testável for um sistema, mais fácil será para os testadores identificarem defeitos e realizar testes de validação. Uma alta pontuação de testabilidade é desejável, uma vez que ajuda a garantir a qualidade e a confiabilidade do software.
WMC: Método Ponderado por Classe — Essa métrica avalia a complexidade de uma classe em relação ao número de métodos que ela contém. Quanto mais métodos, maior o valor de WMC, o que pode indicar uma classe complexa.

APÊNDICE A – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

GABINETE DO REITOR

Comitê de Ética em Pesquisas Envolvendo Seres Humanos - Udesc

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

O(a) senhor(a) está sendo convidado a participar de uma pesquisa mestrado intitulada "SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS DEMANDAS NA ÁREA DE MANUTENÇÃO DE SOFTWARE", que fará uma avaliação, tendo como objetivo auxiliar no planejamento das tarefas da área de manutenção de software, priorizando as tarefas e sugerindo alocações de recurso. Esta pesquisa envolve ambientes virtuais como formulários, uso de aplicativos de chamada e a navegação no *framework* desenvolvido. Serão previamente marcados a data e horário para realização do experimento, utilizando um computador para navegar no *framework*, bem e responder ao questionário e preenchimento de planilhas disponíveis de forma on-line. Estas medidas serão realizadas no seu local de trabalho ou estudo. Também serão realizados capacitações de uso do *framework* antes de sua realização. Não é obrigatório a participação na pesquisa ou a resposta de todas as perguntas.

O(a) Senhor(a) e seu/sua acompanhante não terão despesas e nem serão remunerados pela participação na pesquisa. Todas as despesas decorrentes de sua participação serão ressarcidas. Em caso de danos, decorrentes da pesquisa será garantida a indenização.

Os riscos destes procedimentos serão mínimos por envolver apenas a coleta de informações que visão caracterizar a forma de gestão, execução das tarefa e atividades, bem como a identificação de possíveis melhoras nos processos destas atividades.

A sua identidade será preservada pois cada indivíduo será identificado por um número de registro no banco de dados, com acesso realizada apenas pelo pesquisador e por suas orientadoras. Todos os dados serão apagados após a realização da análise estatística que ocorrerá em, no máximo, 6 meses a partir da data de termino desta avaliação.

Os benefícios e vantagens em participar deste estudo serão em curto prazo permitir a evolução deste projeto de pesquisa e a longo prazo, permitir que a comunidade acadêmica tenha material científico para abordar a priorização de tarefas, além do framework, resultado dessa pesquisa, que será disponibilizado sob licença de uso livre.

A pessoa que estará acompanhando os procedimentos será o pesquisador Jackson Machado (aluno do Programa de Pós Graduação em Computação Aplicada) e por suas orientadoras Dra. Avanilde Kemczinski (CCT/UDESC) e Dra. Rebeca Schroeder Freitas (CCT/UDESC).

O(a) senhor(a) poderá se retirar do estudo a qualquer momento, sem qualquer tipo de constrangimento.

Solicitamos a sua autorização para o uso de seus dados para a produção de artigos técnicos e científicos. A sua privacidade será mantida através da não-identificação do seu nome.

Este termo de consentimento livre e esclarecido é feito em duas vias, sendo que uma delas ficará em poder do pesquisador e outra com o sujeito participante da pesquisa.

NOME DO PESQUISADOR RESPONSÁVEL PARA CONTATO: Jackson Macahdo
NÚMERO DO TELEFONE: (47) 99932-6658
ENDEREÇO: Rua Félix Heinzelmann, 105, Joinville – SC.
E-MAIL: jackson_machado_12@hotmail.com
NOME DA ORIENTADORA RESPONSÁVEL PARA CONTATO: Avanilde Kemczinski
NÚMERO DO TELEFONE: (47) 99197-6799
E-MAIL: avanilde.kemczinski@udesc.br
ASSINATURA DO PESQUISADOR:

Comitê de Ética em Pesquisa Envolvendo Seres Humanos – CEPSH/UDESC
Av. Madre Benvenuta, 2007 – Itacorubi – Florianópolis – SC - 88035-901
Fone/Fax: (48) 3664-8084 / (48) 3664-7881 - E-mail: cep.udesc@gmail.com
CONEP- Comissão Nacional de Ética em Pesquisa
SRTV 701, Via W 5 Norte – lote D - Edifício PO 700, 3º andar – Asa Norte - Brasília-DF - 70719-040
Fone: (61) 3315-5878/ 5879 – E-mail: conep@saude.gov.br

TERMO DE CONSENTIMENTO

Declaro que fui informado sobre todos os procedimentos da pesquisa e, que recebi de forma clara e objetiva todas as explicações pertinentes ao projeto e, que todos os dados a meu respeito serão sigilosos. Eu compreendo que neste estudo, as medições dos experimentos/procedimentos de tratamento serão feitas em mim, e que fui informado que posso me retirar do estudo a qualquer momento.

Nome por extenso _____

Assinatura _____ Local: _____ Data: ___/___/___ .

Avenida Madre Benvenuta, 2007, Itacorubi, CEP 88035-901, Florianópolis, SC, Brasil.
Telefone/Fax: (48) 3664-8084 / (48) 3664-7881 - E-mail: cep.udesc@gmail.com
CONEP- Comissão Nacional de Ética em Pesquisa
SRTV 701, Via W 5 Norte – Lote D - Edifício PO 700, 3º andar – Asa Norte - Brasília-DF - 70719-040
Fone: (61) 3315-5878/ 5879 – E-mail: conep@saude.gov.br

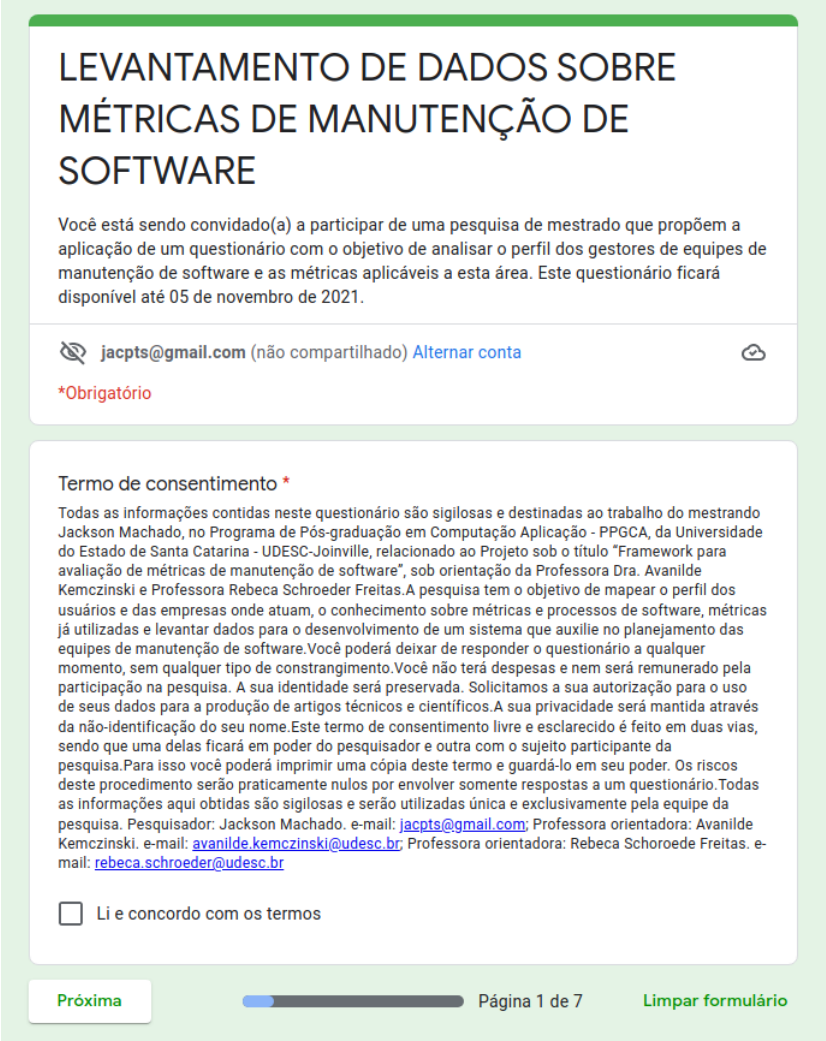
APÊNDICE B – ARTIGOS ENCONTRADOS NA REVISÃO SISTEMÁTICA

1	(AHN et al., 2003)
2	(AL-JAMIMI; AHMED, 2012)
3	(BAKOTA et al., 2011)
4	(BHATT et al., 2006)
5	(BURROWS et al., 2010)
6	(BöRSTLER; CASPERSEN; NORDSTRöM, 2015)
7	(ÇALIKLI; STARON; MEDING, 2018)
8	(CANFORA et al., 2005)
9	(CHOUDHARI; SUMAN, 2012)
10	(CONCAS et al., 2010)
11	(COUTO et al., 2018)
12	(DAGPINAR; WEBER, 2003)
13	(DAHAB; PORRAS; MAAG, 2018)
14	(DANTAS; GARCIA; WHITTLE, 2012)
15	(EL-SHARKAWY; KRAFCZYK; SCHMID, 2020)
16	(FIORAVANTI; NESI; STORTONI, 1999)
17	(GARCÍA; RUIZ; PIATTINI, 2004)
18	(GIL; GOLDSTEIN; MOSHKOVICH, 2011)
19	(GIL; GOLDSTEIN; MOSHKOVICH, 2012)
20	(GORDEA; ZANKER, 2006)
21	(HAYES; MOHAMED; GAO, 2003)
22	(HAYES; PATEL; ZHAO, 2004)
23	(HEGEDüS et al., 2013)
24	(HOVSEPYAN et al., 2010)
25	(JAIN; TARWANI; CHUG, 2016)
26	(JøRGENSEN; SJøBERG, 2002)
27	(KULKARNI; KALSHETTY; ARDE, 2010)
28	(LEWIS; HENRY, 1990)
29	(LI; HENRY, 1993)
30	(MAL; RAJNISH, 2014)
31	(MAMUN; BERGER; HANSSON, 2017)
32	(OSTBERG; WAGNER, 2014)
33	(RAZANI; KEYVANPOUR, 2019)
34	(SABOE, 2001)
35	(SAIFAN; AL-RABADI, 2017)
36	(SANGWAN; SINGH, 2013)
37	(SCHNAPPINGER et al., 2019)
38	(SELVARANI; NAIR; PRASAD, 2009)
39	(SHARMA; BHATIA; GROVER, 2008)
40	(SHELDON; JERATH; CHUNG, 2002)

41	(HUANG; LAI, 2003)
42	(SURESH; PATI; RATH, 2012)
43	(THONGMAK; MUENCHAISRI, 2009)
44	(ZHANG et al., 2013)

Fonte: Elaborado pelo autor (2023).

APÊNDICE C – QUESTIONÁRIO DA PESQUISA EXPLORATÓRIA



LEVANTAMENTO DE DADOS SOBRE MÉTRICAS DE MANUTENÇÃO DE SOFTWARE

Você está sendo convidado(a) a participar de uma pesquisa de mestrado que propõem a aplicação de um questionário com o objetivo de analisar o perfil dos gestores de equipes de manutenção de software e as métricas aplicáveis a esta área. Este questionário ficará disponível até 05 de novembro de 2021.

jacpts@gmail.com (não compartilhado) Alternar conta

*Obrigatório

Termo de consentimento *

Todas as informações contidas neste questionário são sigilosas e destinadas ao trabalho do mestrando Jackson Machado, no Programa de Pós-graduação em Computação Aplicação - PPGCA, da Universidade do Estado de Santa Catarina - UDESC-Joinville, relacionado ao Projeto sob o título "Framework para avaliação de métricas de manutenção de software", sob orientação da Professora Dra. Avanilde Kemczinski e Professora Rebeca Schroeder Freitas.A pesquisa tem o objetivo de mapear o perfil dos usuários e das empresas onde atuam, o conhecimento sobre métricas e processos de software, métricas já utilizadas e levantar dados para o desenvolvimento de um sistema que auxilie no planejamento das equipes de manutenção de software.Você poderá deixar de responder o questionário a qualquer momento, sem qualquer tipo de constrangimento.Você não terá despesas e nem será remunerado pela participação na pesquisa. A sua identidade será preservada. Solicitamos a sua autorização para o uso de seus dados para a produção de artigos técnicos e científicos.A sua privacidade será mantida através da não-identificação do seu nome.Este termo de consentimento livre e esclarecido é feito em duas vias, sendo que uma delas ficará em poder do pesquisador e outra com o sujeito participante da pesquisa.Para isso você poderá imprimir uma cópia deste termo e guardá-lo em seu poder. Os riscos deste procedimento serão praticamente nulos por envolver somente respostas a um questionário.Todas as informações aqui obtidas são sigilosas e serão utilizadas única e exclusivamente pela equipe da pesquisa. Pesquisador: Jackson Machado. e-mail: jacpts@gmail.com; Professora orientadora: Avanilde Kemczinski. e-mail: avanilde.kemczinski@udesc.br; Professora orientadora: Rebeca Schoroede Freitas. e-mail: rebeca.schroeder@udesc.br

Li e concordo com os termos

Próxima Página 1 de 7 Limpar formulário

Perfil do Participante

Qual a sua idade *

- ◯ 18-24 anos
- ◯ 25-34 anos
- ◯ 35-44 anos
- ◯ 45-54 anos
- ◯ 55-64 anos
- ◯ 65 anos ou mais
- ◯ Prefiro não informar

Qual a sua cidade/região de origem? *

- ◯ Joinville/SC
- ◯ Jaraguá do Sul/SC
- ◯ Blumenau/SC
- ◯ Florianópolis/SC
- ◯ Chapecó/SC
- ◯ Itajaí/SC
- ◯ Balneário Camboriú/SC
- ◯ Criciúma/SC
- ◯ Porto Alegre/RS
- ◯ Curitiba/PR
- ◯ São Paulo/SP
- ◯ Outro: _____

Quando foi o seu primeiro contato com a área de manutenção de software? *

- ◯ A menos de 1 ano
- ◯ 1 a 4 anos
- ◯ 5 a 9 anos
- ◯ 10 a 14 anos
- ◯ 15 a 19 anos
- ◯ 20 a 24 anos
- ◯ 25 a 29 anos
- ◯ 30 a 39 anos
- ◯ 40 a 49 anos
- ◯ Mais de 50 anos

A quanto tempo você realiza a gestão de uma área de manutenção de software? *

- ◯ A menos de 1 ano
- ◯ 1 a 4 anos
- ◯ 5 a 9 anos
- ◯ 10 a 14 anos
- ◯ 15 a 19 anos
- ◯ 20 a 24 anos
- ◯ 25 a 29 anos
- ◯ 30 a 39 anos
- ◯ 40 a 49 anos
- ◯ Mais de 50 anos

A quantos anos você considera que está fora da área técnica? (Considera-se fora da área técnica aqueles que não praticam mais atividades de codificação, testes e afins) *

- ◯ A menos de 1 ano
- ◯ 1 a 4 anos
- ◯ 5 a 9 anos
- ◯ 10 a 14 anos
- ◯ 15 a 19 anos
- ◯ 20 a 24 anos
- ◯ 25 a 29 anos
- ◯ 30 a 39 anos
- ◯ 40 a 49 anos
- ◯ Mais de 50 anos

Atualmente qual a nomenclatura do seu cargo? *

- ◯ Desenvolvedor
- ◯ Líder Técnico (Tech Lead)
- ◯ Dono do Produto (Product Owner)
- ◯ Scrum Master
- ◯ Gerente de Produto (Product Manager)
- ◯ Coordenador de Software
- ◯ Engenheiro de Software
- ◯ Outro: _____

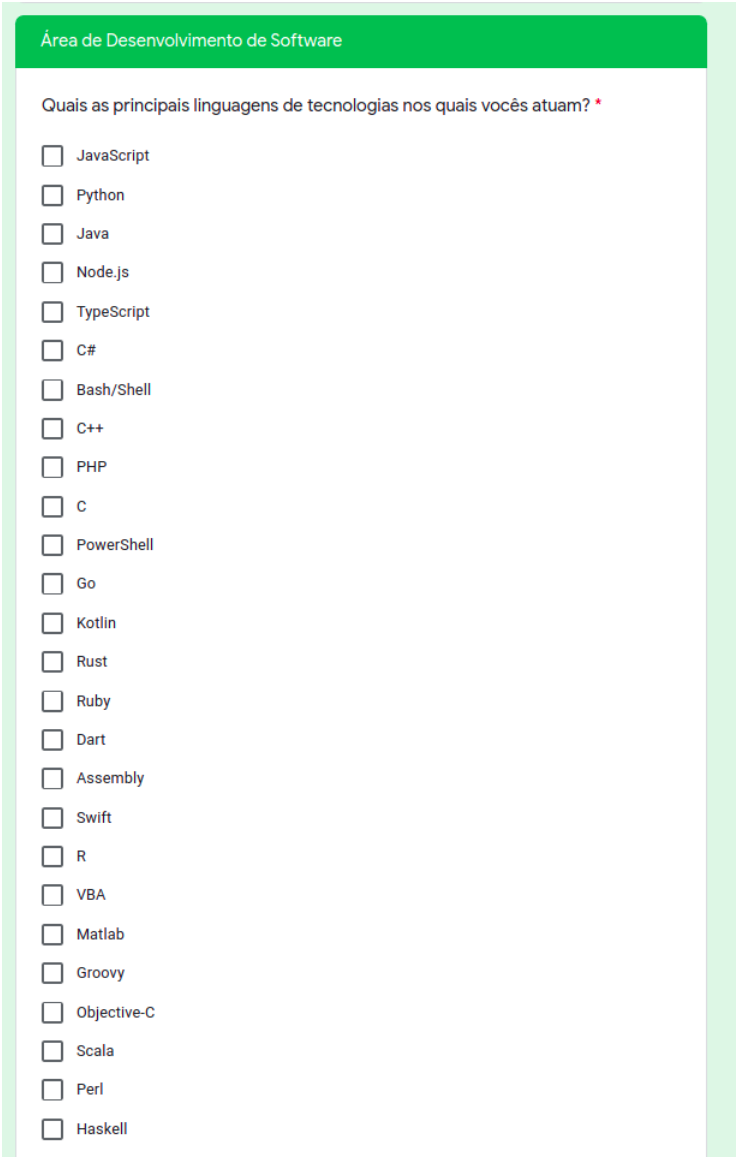
Qual o seu grau de instrução? *

- ◯ Escola primária
- ◯ Escola secundária
- ◯ Algum estudo de faculdade/ universidade sem obter um diploma
- ◯ Tecnólogo
- ◯ Bacharelado
- ◯ Pós-Graduação (Lato Sensu)
- ◯ Pós-Graduação (Stricto Sensu)/Mestrado
- ◯ Doutorado
- ◯ Pós-doutorado
- ◯ Outro: _____

Costumeiramente você procura atualização profissional através de quais meios? *

- ☐ Escola
- ☐ Livros/mídia física
- ☐ Cursos ou certificação online
- ☐ Fórum Online
- ☐ Amigo ou familiar
- ☐ Bootcamp/Barcamp/etc
- ☐ Outros recursos online (vídeos, blogs, etc)

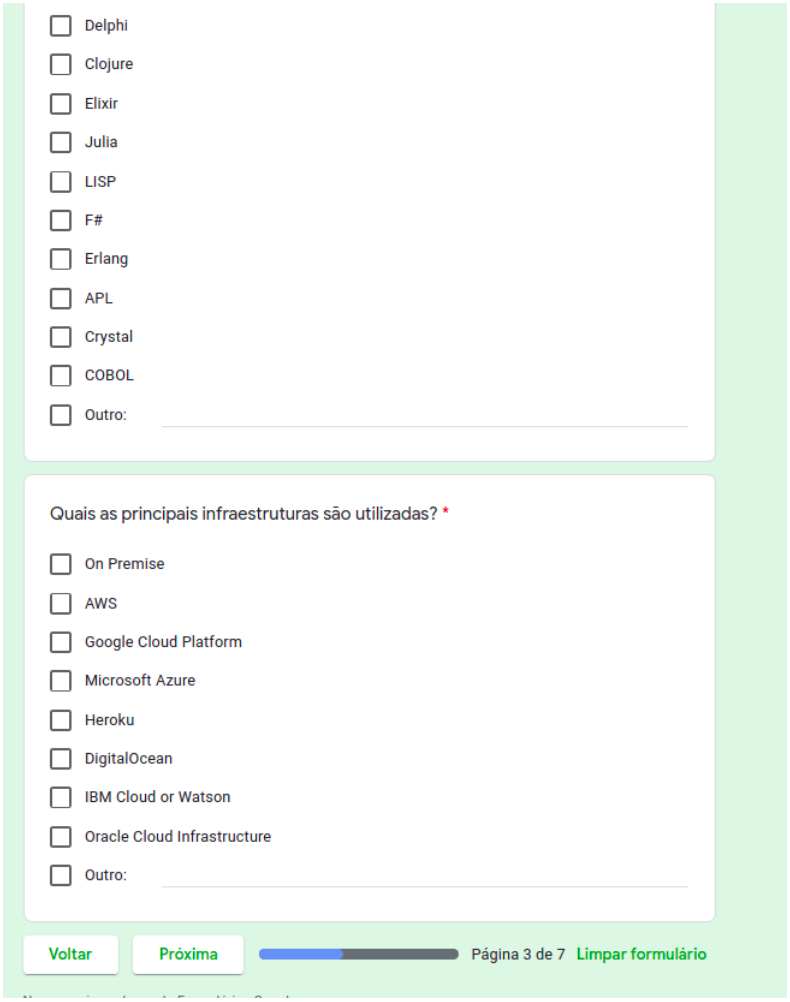
Voltar Próxima Página 2 de 7 Limpar formulário



Área de Desenvolvimento de Software

Quais as principais linguagens de tecnologias nos quais vocês atuam? *

- [ ] JavaScript
- [ ] Python
- [ ] Java
- [ ] Node.js
- [ ] TypeScript
- [ ] C#
- [ ] Bash/Shell
- [ ] C++
- [ ] PHP
- [ ] C
- [ ] PowerShell
- [ ] Go
- [ ] Kotlin
- [ ] Rust
- [ ] Ruby
- [ ] Dart
- [ ] Assembly
- [ ] Swift
- [ ] R
- [ ] VBA
- [ ] Matlab
- [ ] Groovy
- [ ] Objective-C
- [ ] Scala
- [ ] Perl
- [ ] Haskell


Delphi
Clojure
Elixir
Julia
LISP
F#
Erlang
APL
Crystal
COBOL
Outro:
Quais as principais infraestruturas são utilizadas? *
On Premise
AWS
Google Cloud Platform
Microsoft Azure
Heroku
DigitalOcean
IBM Cloud or Watson
Oracle Cloud Infrastructure
Outro:
Voltar
Próxima
Página 3 de 7
Limpar formulário

Organizacional

Qual a sua situação de emprego atual? *

- ◯ Empregado em tempo integral
- ◯ Contratante independente, freelancer ou autônomo
- ◯ Empregado em tempo parcial
- ◯ Não empregado, mas procurando trabalho
- ◯ Não empregado e não procurando trabalho
- ◯ Aposentado
- ◯ Eu prefiro não dizer

Qual o tamanho da empresa na qual você trabalha? *

- ◯ Só eu - sou freelancer, único proprietário, etc.
- ◯ 2 a 9 funcionários
- ◯ 10 a 19 funcionários
- ◯ 20 a 99 funcionários
- ◯ 100 a 499 funcionários
- ◯ 500 a 999 funcionários
- ◯ 1.000 a 4.999 funcionários
- ◯ 5.000 a 9.999 funcionários
- ◯ 10.000 ou mais funcionários
- ◯ Não sei informar

Qual o modelo de ciclo de vida de software aplicado em sua empresa? *

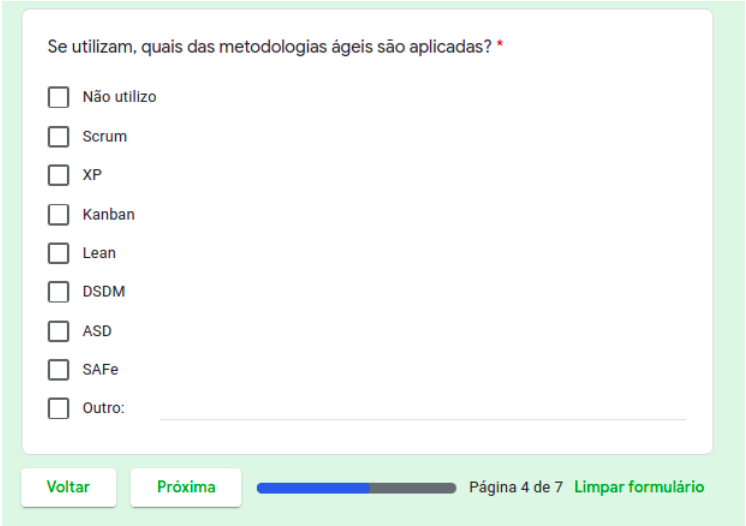
- ◯ Cascata
- ◯ Evolutivo
- ◯ Incremental
- ◯ Espiral

Qual o seu grau de familiaridade com as metodologias ágeis? Sendo 1 pouco familiar e 7 muito familiar *

1	2	3	4	5	6	7
◯	◯	◯	◯	◯	◯	◯

Qual o grau de aplicação da sua companhia com metodologias ágeis? Sendo 1 pouco familiar e 7 naturalmente aplicável *

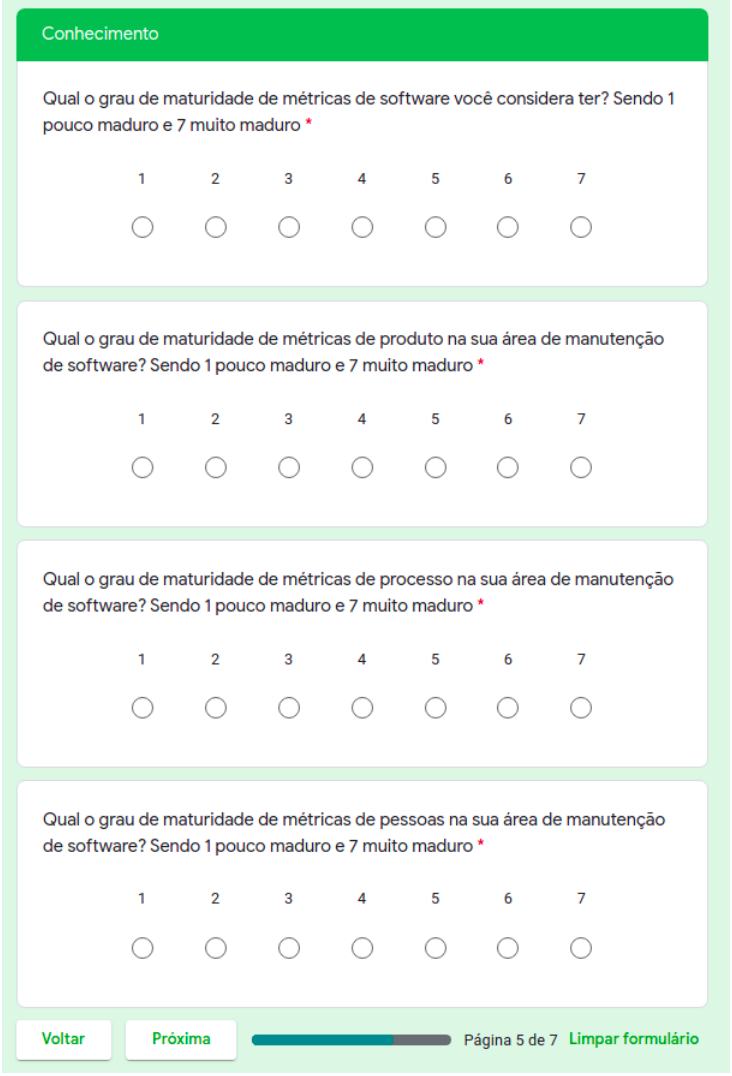
1	2	3	4	5	6	7
◯	◯	◯	◯	◯	◯	◯



Se utilizam, quais das metodologias ágeis são aplicadas? *

- [ ] Não utilizo
- [ ] Scrum
- [ ] XP
- [ ] Kanban
- [ ] Lean
- [ ] DSDM
- [ ] ASD
- [ ] SAFe
- [ ] Outro:

Voltar Próxima Página 4 de 7 Limpar formulário



Conhecimento

Qual o grau de maturidade de métricas de software você considera ter? Sendo 1 pouco maduro e 7 muito maduro *

1 2 3 4 5 6 7

Qual o grau de maturidade de métricas de produto na sua área de manutenção de software? Sendo 1 pouco maduro e 7 muito maduro *

1 2 3 4 5 6 7

Qual o grau de maturidade de métricas de processo na sua área de manutenção de software? Sendo 1 pouco maduro e 7 muito maduro *

1 2 3 4 5 6 7

Qual o grau de maturidade de métricas de pessoas na sua área de manutenção de software? Sendo 1 pouco maduro e 7 muito maduro *

1 2 3 4 5 6 7

Voltar Próxima Página 5 de 7 Limpar formulário

Métricas

Quais das métricas abaixo são aplicáveis hoje ao produto de sua área de manutenção de software? *

- [ ] LoC - Linhas Lógicas de Código
- [ ] WMC - Método ponderado por classe
- [ ] DIT - Profundidade na árvore de herança
- [ ] CBO - Acoplamento entre classes de objetos
- [ ] RFC - Respostas por classe
- [ ] NOC - Número de filhos
- [ ] LCOM - Falta de coesão de métodos
- [ ] N - Tamanho do Programa
- [ ] CC - Complexidade da Classe
- [ ] NOM - Número de Métodos
- [ ] NOA - Número de Atributos
- [ ] Fan-In - Número de Entradas
- [ ] Fan-Out - Número de Saídas
- [ ] McCC - Complexidade ciclomática de McCabe
- [ ] HEFF - Métricas de Halsted
- [ ] V - Volume
- [ ] FP - Ponto de Função
- [ ] COCOMO
- [ ] HP - Pontos de História
- [ ] CP - Pontos de Caso de Uso
- [ ] Outro:

Quais das métricas abaixo são aplicáveis hoje ao processo da sua área de manutenção de software? *

- [ ] MI - Índice de Manutenção
- [ ] NPR - Número de funções que participam do processo
- [ ] NDA - Número de dependências de precedência entre atividades
- [ ] NDWP - Número de dependências entre produtos de trabalho e atividades
- [ ] NDWPIn - Número de dependências de entrada dos produtos de trabalho com as atividades no processo
- [ ] NDWPOut - Número de dependências de saída dos produtos de trabalho com as atividades no processo
- [ ] SoM - Tamanho da Manutenção
- [ ] DoM - Duração da manutenção
- [ ] BDX - Complexidade de implantação do sistema
- [ ] DOC - Atualização da documentação
- [ ] BDS - Disponibilidade de informações sobre estatísticas de defeitos no início do projeto
- [ ] BFQ - Frequência de falha no sistema na iniciação
- [ ] DD - densidade de defeitos - Número de defeitos por linhas de código
- [ ] SES - Conformidade com os padrões de engenharia de software
- [ ] TST - Testabilidade
- [ ] PM - Manutenção Percebida
- [ ] DM - modificações de defeitos - O número de modificações identificadas para cada defeito por função
- [ ] Tasks - Número de Tarefas
- [ ] B&W - A legibilidade do software (Buse and Weimer)
- [ ] RWPA - Proporção de produtos de trabalho e atividades
- [ ] Outro: _____

Atualmente é utilizado algum dos seguintes padrões para certificação dos processos? *

- [ ] Não aplicável
- [ ] ISO/IEC 12207
- [ ] CMMI - Modelo de Maturidade em Capacitação - Integração (anteriormente CMM)
- [ ] ISO 9000
- [ ] ISO/IEC 15504 (anteriormente SPICE)
- [ ] MR-MPS
- [ ] IBM Rational Unified Process

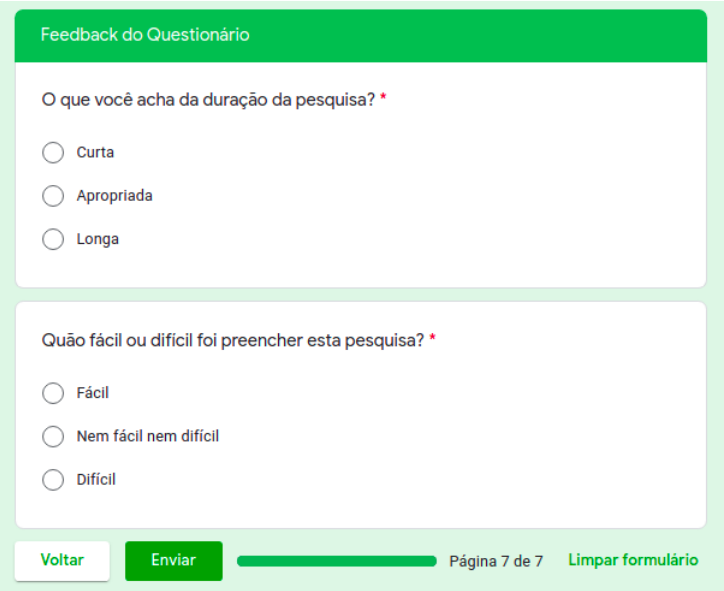
Quais das métricas abaixo são aplicáveis hoje as pessoas da sua área de manutenção de software? *

- [ ] E - Esforço
- [ ] D - Dificuldade
- [ ] BDD - Complexidade de domínio do sistema
- [ ] CONF - Confiança na resolução de tarefas
- [ ] DEVMSYS - Experiência de desenvolvimento de módulo
- [ ] DEVSYS - Experiência de desenvolvimento da aplicação
- [ ] ESS - Experiência com desenvolvimento de sistemas
- [ ] FPL - Familiaridade com a linguagem
- [ ] KAD - Conhecimento do domínio da aplicação
- [ ] MEXPAPP - Experiência de manutenção na aplicação
- [ ] MEXPTOT - Experiência em manutenção de software
- [ ] Outro: _____

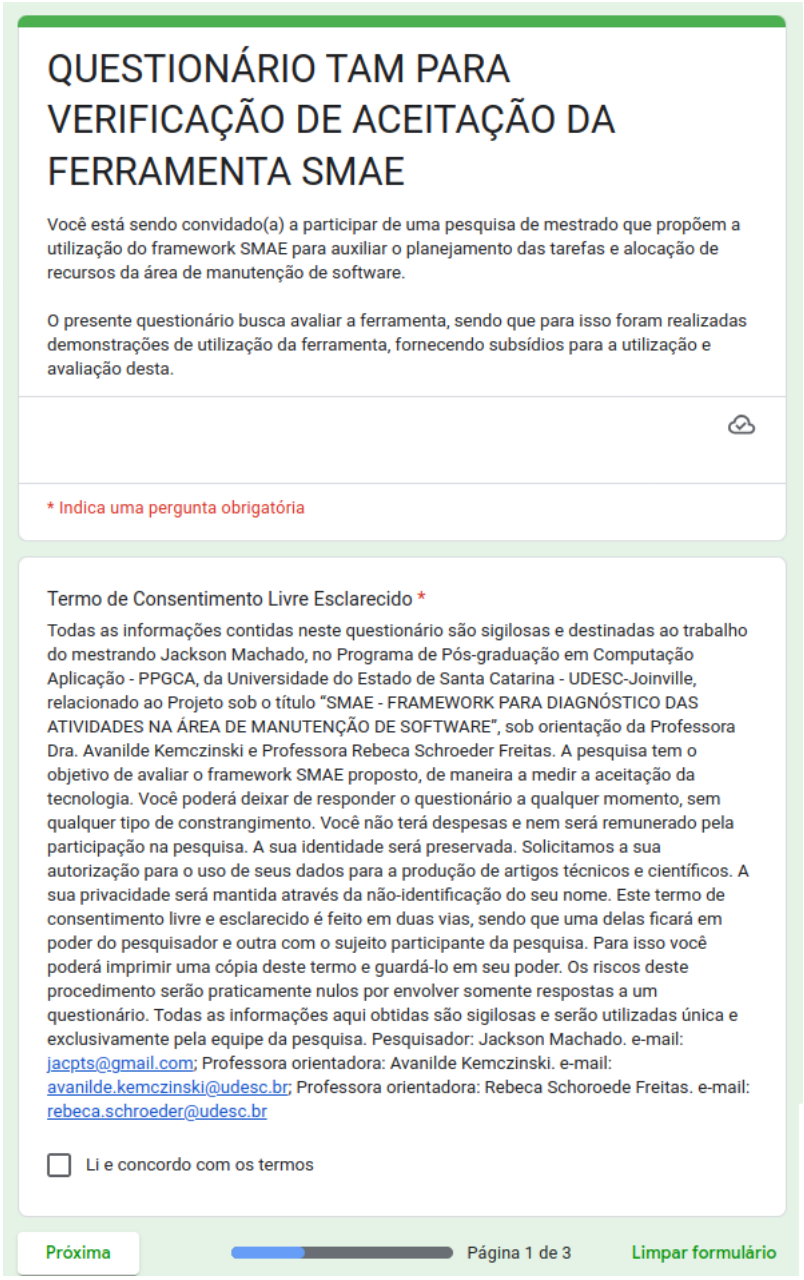
Há ainda alguma métrica não citada que não se enquadre em alguma das categorias acima que você gostaria de pontuar?

Sua resposta

Voltar Próxima Página 6 de 7 Limpar formulário


Feedback do Questionário
O que você acha da duração da pesquisa? *
Curta
Apropriada
Longa
Quão fácil ou difícil foi preencher esta pesquisa? *
Fácil
Nem fácil nem difícil
Difícil
Voltar
Enviar
Página 7 de 7
Limpar formulário

APÊNDICE D – QUESTIONÁRIO TAM PARA VERIFICAÇÃO DE ACEITAÇÃO DA FERRAMENTA SMAE



QUESTIONÁRIO TAM PARA VERIFICAÇÃO DE ACEITAÇÃO DA FERRAMENTA SMAE

Você está sendo convidado(a) a participar de uma pesquisa de mestrado que propõem a utilização do framework SMAE para auxiliar o planejamento das tarefas e alocação de recursos da área de manutenção de software.

O presente questionário busca avaliar a ferramenta, sendo que para isso foram realizadas demonstrações de utilização da ferramenta, fornecendo subsídios para a utilização e avaliação desta.

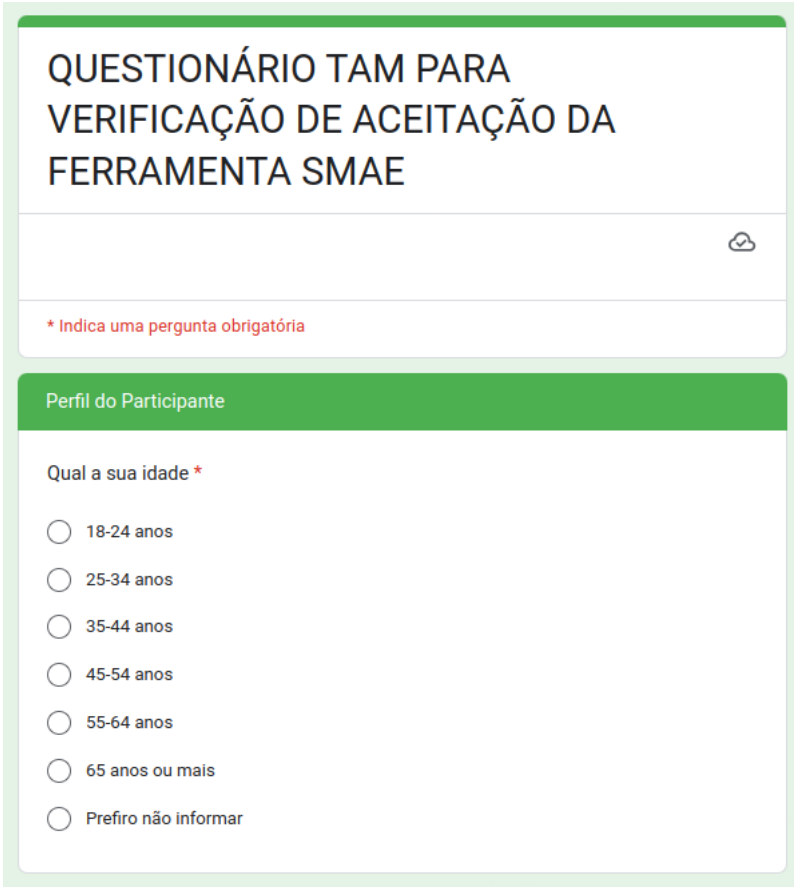
* Indica uma pergunta obrigatória

Termo de Consentimento Livre Esclarecido *

Todas as informações contidas neste questionário são sigilosas e destinadas ao trabalho do mestrando Jackson Machado, no Programa de Pós-graduação em Computação Aplicação - PPGCA, da Universidade do Estado de Santa Catarina - UDESC-Joinville, relacionado ao Projeto sob o título "SMAE - FRAMEWORK PARA DIAGNÓSTICO DAS ATIVIDADES NA ÁREA DE MANUTENÇÃO DE SOFTWARE", sob orientação da Professora Dra. Avanilde Kemczinski e Professora Rebeca Schroeder Freitas. A pesquisa tem o objetivo de avaliar o framework SMAE proposto, de maneira a medir a aceitação da tecnologia. Você poderá deixar de responder o questionário a qualquer momento, sem qualquer tipo de constrangimento. Você não terá despesas e nem será remunerado pela participação na pesquisa. A sua identidade será preservada. Solicitamos a sua autorização para o uso de seus dados para a produção de artigos técnicos e científicos. A sua privacidade será mantida através da não-identificação do seu nome. Este termo de consentimento livre e esclarecido é feito em duas vias, sendo que uma delas ficará em poder do pesquisador e outra com o sujeito participante da pesquisa. Para isso você poderá imprimir uma cópia deste termo e guardá-lo em seu poder. Os riscos deste procedimento serão praticamente nulos por envolver somente respostas a um questionário. Todas as informações aqui obtidas são sigilosas e serão utilizadas única e exclusivamente pela equipe da pesquisa. Pesquisador: Jackson Machado. e-mail: jacpts@gmail.com; Professora orientadora: Avanilde Kemczinski. e-mail: avanilde.kemczinski@udesc.br; Professora orientadora: Rebeca Schoroede Freitas. e-mail: rebeca.schroeder@udesc.br

☐ Li e concordo com os termos

Próxima Página 1 de 3 Limpar formulário



QUESTIONÁRIO TAM PARA VERIFICAÇÃO DE ACEITAÇÃO DA FERRAMENTA SMAE

* Indica uma pergunta obrigatória

Perfil do Participante

Qual a sua idade *

- 18-24 anos
- 25-34 anos
- 35-44 anos
- 45-54 anos
- 55-64 anos
- 65 anos ou mais
- Prefiro não informar

Quando foi o seu primeiro contato com a área de manutenção de software? *

- ◯ A menos de 1 ano
- ◯ 1 a 4 anos
- ◯ 5 a 9 anos
- ◯ 10 a 14 anos
- ◯ 15 a 19 anos
- ◯ 20 a 24 anos
- ◯ 25 a 29 anos
- ◯ 30 a 39 anos
- ◯ 40 a 49 anos
- ◯ Mais de 49 anos

A quanto tempo você realiza a gestão de uma área de manutenção de software? *

- ◯ A menos de 1 ano
- ◯ 1 a 4 anos
- ◯ 5 a 9 anos
- ◯ 10 a 14 anos
- ◯ 15 a 19 anos
- ◯ 20 a 24 anos
- ◯ 25 a 29 anos
- ◯ 30 a 39 anos
- ◯ 40 a 49 anos
- ◯ Mais de 49 anos

Atualmente qual a nomenclatura do seu cargo? *

- ◯ Desenvolvedor
- ◯ Líder Técnico (Tech Lead)
- ◯ Dono do Produto (Product Owner)
- ◯ Scrum Master
- ◯ Gerente de Produto (Product Manager)
- ◯ Coordenador de Software
- ◯ Engenheiro de Software
- ◯ Outro: _____

Qual o seu grau de instrução? *

- ◯ Escola primária
- ◯ Escola secundária
- ◯ Algum estudo de faculdade/ universidade sem obter um diploma
- ◯ Tecnólogo
- ◯ Bacharelado
- ◯ Pós-Graduação (Lato Sensu)
- ◯ Pós-Graduação (Stricto Sensu)/Mestrado
- ◯ Doutorado
- ◯ Pós-doutorado
- ◯ Outro: _____

Voltar Próxima Página 2 de 3 Limpar formulário

QUESTIONÁRIO TAM PARA VERIFICAÇÃO DE ACEITAÇÃO DA FERRAMENTA SMAE

* Indica uma pergunta obrigatória

Sobre a ferramenta SMAE

A questões abaixo devem ser respondidas visando avaliar a ferramenta SMAE. O valor mais baixo (1 - um) corresponde a discordo totalmente e o valor mais alto (5 - cinco) corresponde a concordo totalmente.

Para tanto, foi preparado um vídeo de demonstração sobre a ferramenta.

Demonstração SMAE

Utilidade Percebida *

	1	2	3	4	5	NA
As demonstrações de uso foram suficientes para compreender as funcionalidades da ferramenta	◯	◯	◯	◯	◯	◯
Usar a ferramenta melhora o meu desempenho durante o planejamento das tarefas de manutenção de software	◯	◯	◯	◯	◯	◯
Usar a ferramenta aumentaria a minha produtividade como gestor da área de manutenção de software	◯	◯	◯	◯	◯	◯
A ferramenta é útil para determinar a alocação de pessoas nas tarefas de manutenção de software	◯	◯	◯	◯	◯	◯

Facilidade de Uso *

	1	2	3	4	5	NA
Com a demonstração, foi fácil entender a ferramenta	◯	◯	◯	◯	◯	◯
Não há dificuldades em localizar as ações a serem realizadas na ferramenta	◯	◯	◯	◯	◯	◯
É fácil a utilização da ferramenta para meu apoio nas atividades de planejamento das tarefas de manutenção de software	◯	◯	◯	◯	◯	◯
A ordem de utilização da ferramenta está disposta de forma lógica, facilitando o uso	◯	◯	◯	◯	◯	◯
A disposição do Grau de Tarefa está de fácil localização e sua ordenação é útil	◯	◯	◯	◯	◯	◯

Intenção de Uso *

	1	2	3	4	5	NA
Eu utilizaria a ferramenta se disponível em minha rotina de trabalho	◯	◯	◯	◯	◯	◯
Minha organização aceitaria a utilização da ferramenta como ferramenta de trabalho	◯	◯	◯	◯	◯	◯

Sugestões, críticas e melhorias *

Sua resposta

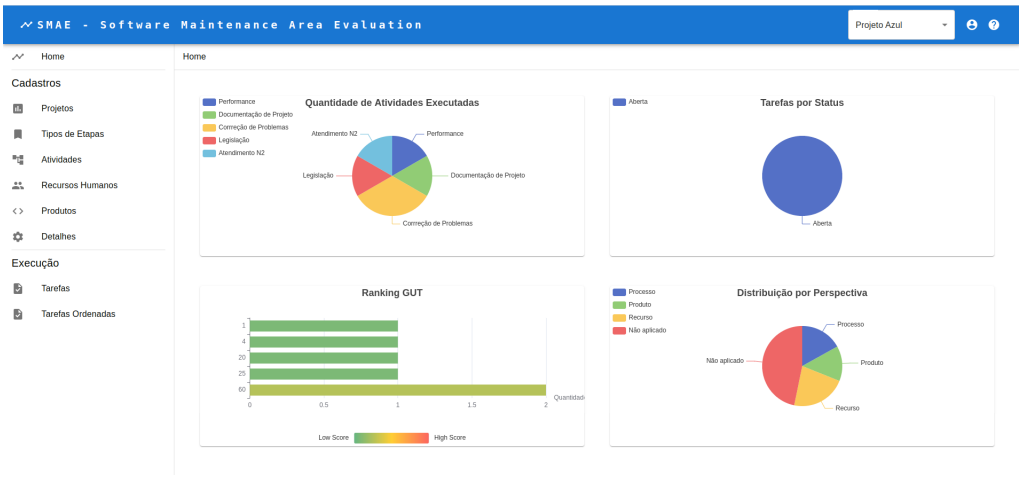
Obrigado!

Obrigado(a) por responder ao nosso questionário. Sua participação é de extrema importância para o avanço deste estudo. Suas respostas nos ajudarão a compreender melhor a importância do *framework* e também a sua maturidade. Não se esqueça de clicar em **Enviar** para registrar suas respostas. Pesquisador: Jackson Machado. e-mail: jacpts@gmail.com; Professora orientadora: Avanilde Kemczinski. e-mail: avanilde.kemczinski@udesc.br; Professora orientadora: Rebeca Schoroede Freitas. e-mail: rebeca.schroeder@udesc.br

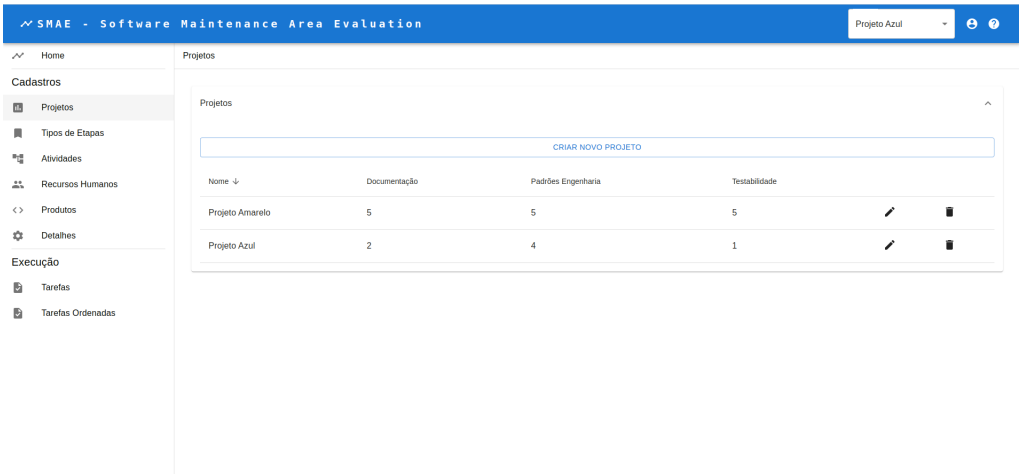
Voltar Enviar Página 3 de 3 Limpar formulário

APÊNDICE E – FERRAMENTA PARA AVALIAÇÃO DO SMAE

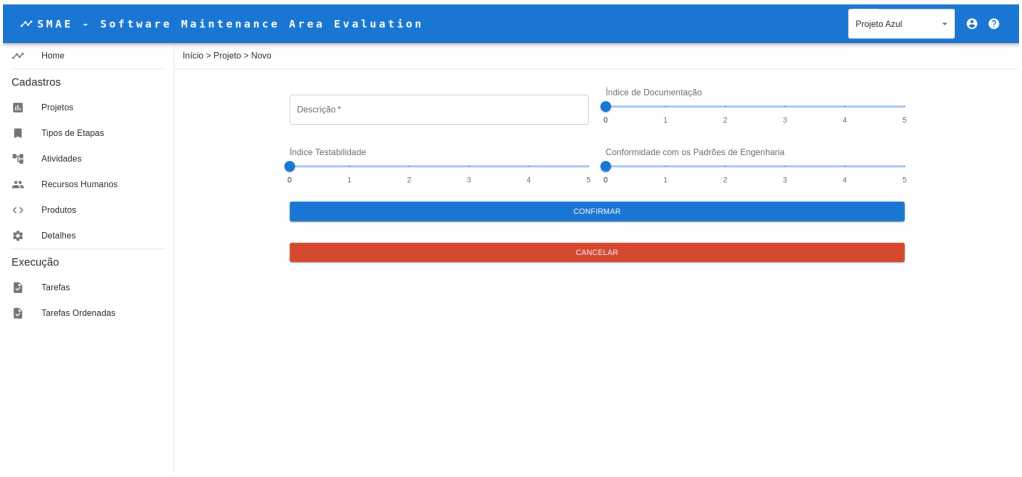
Dashboard Inicial



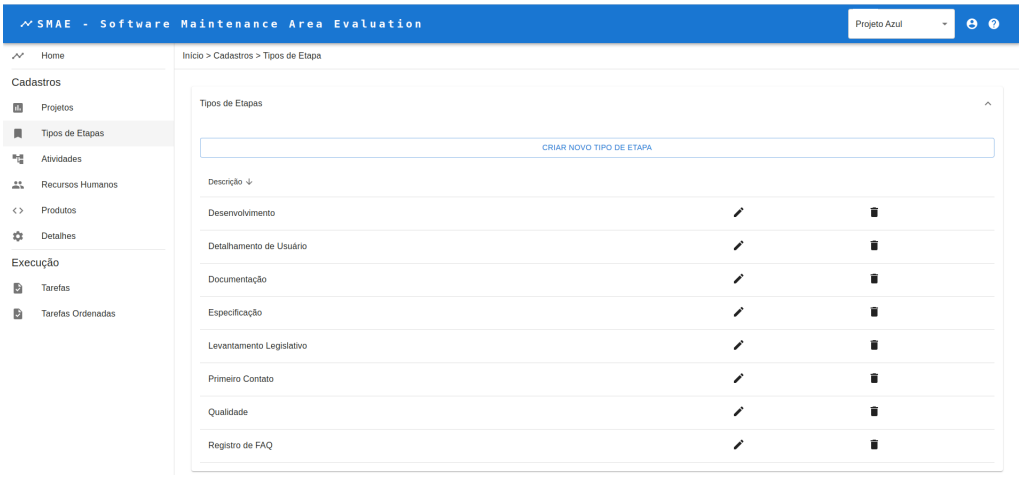
Lista de Projetos



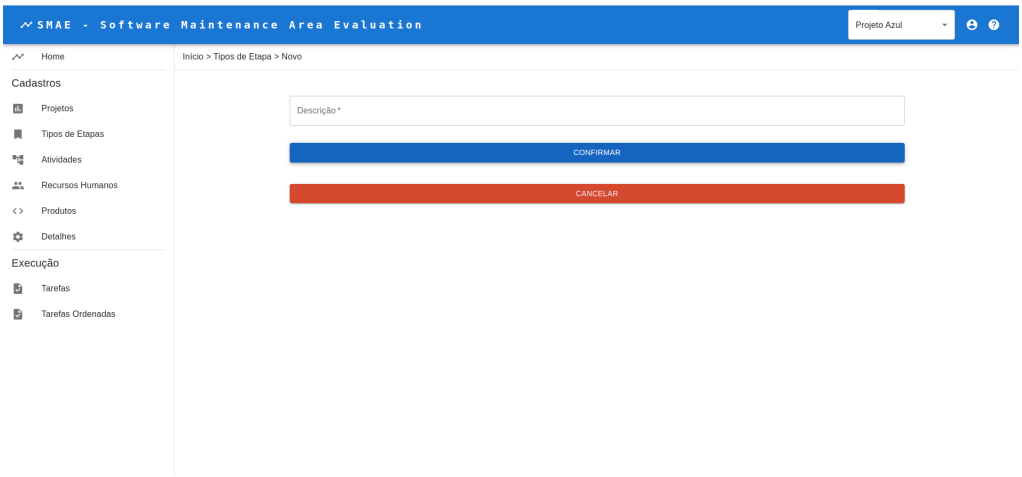
Criação de Projetos



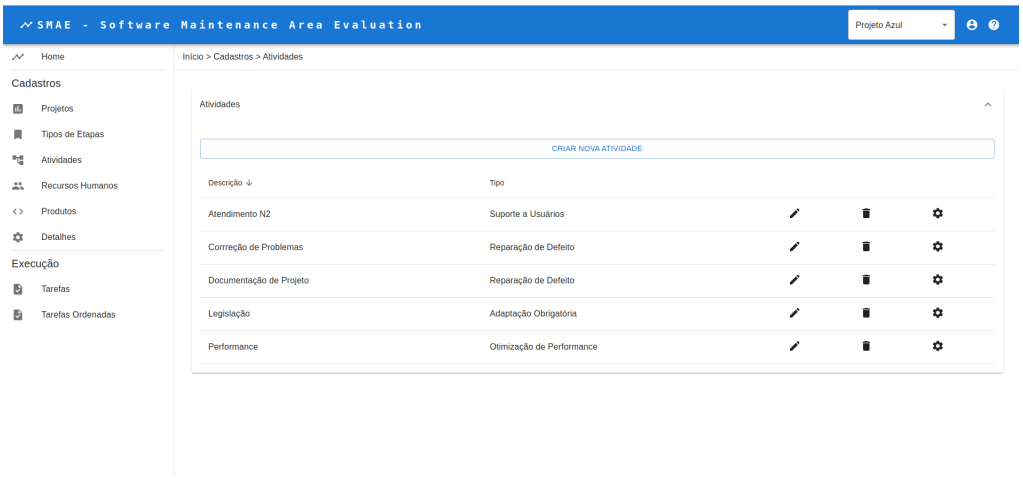
Lista de Etapas



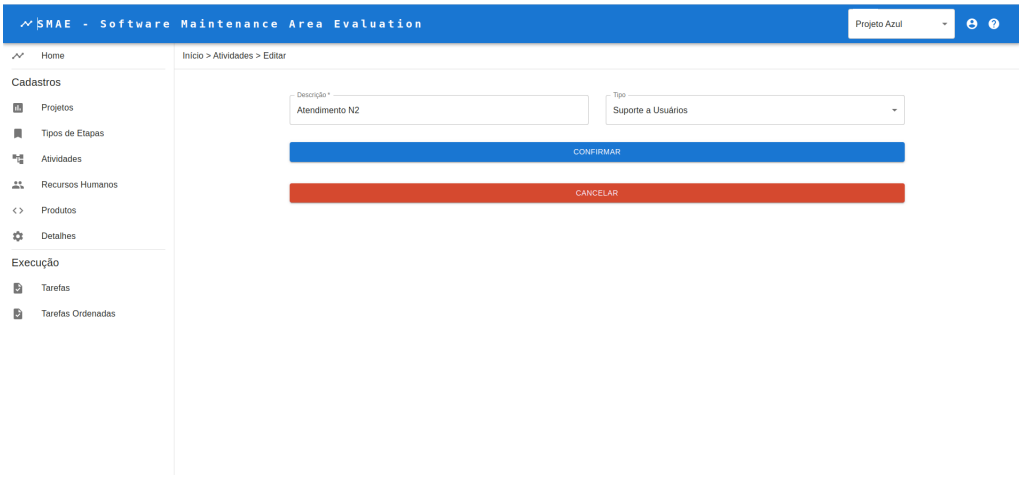
Criação de Etapas



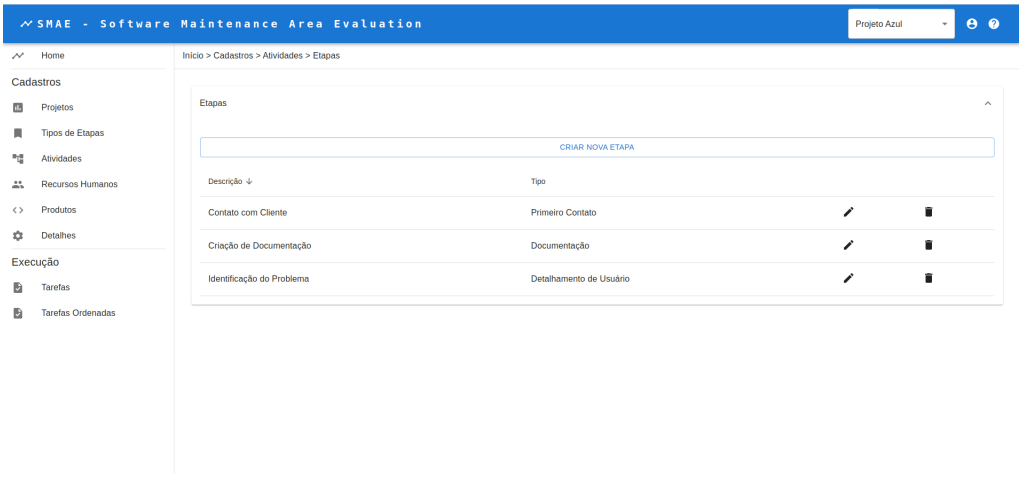
Lista de Atividades



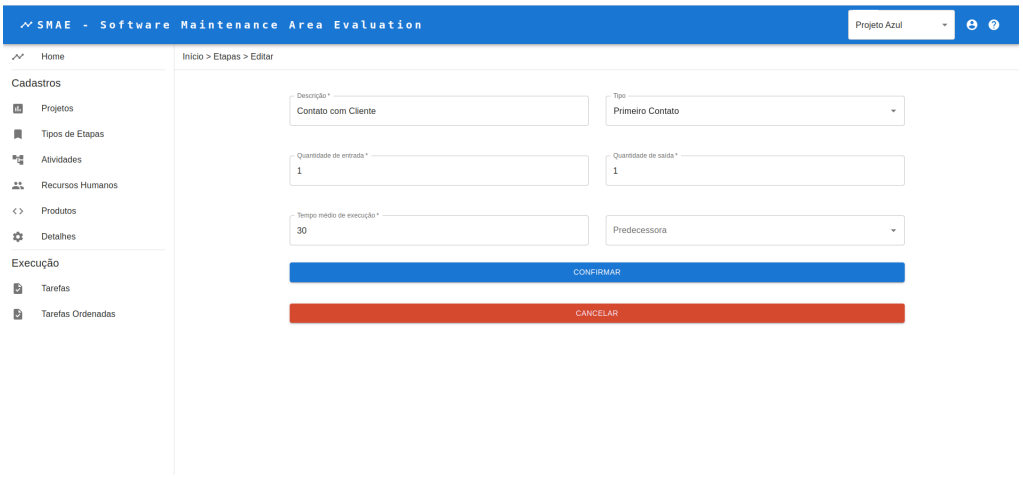
Criação de Atividades



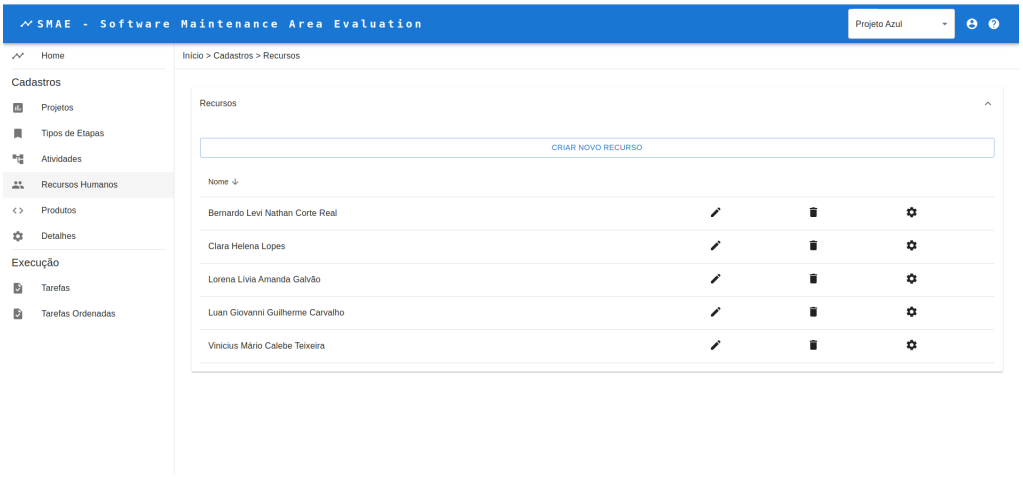
Lista de Etapas



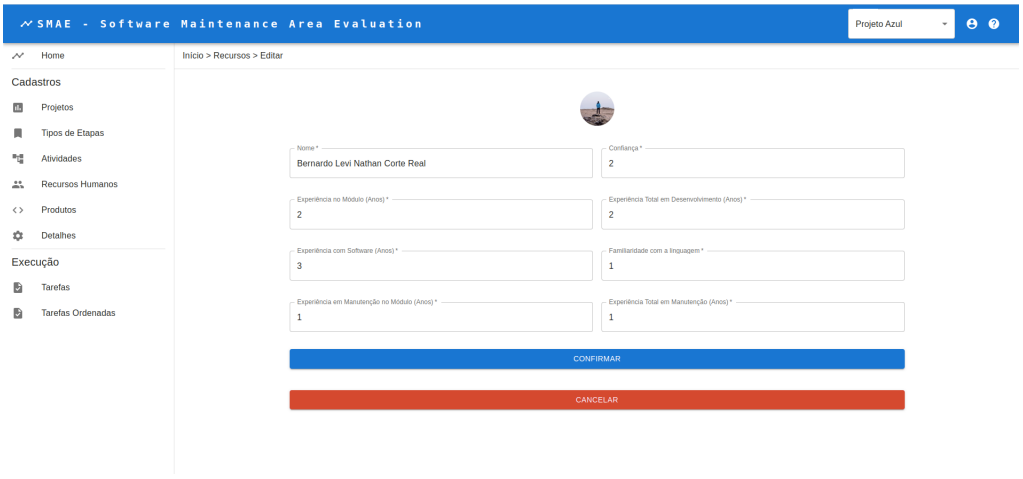
Criação de Etapas



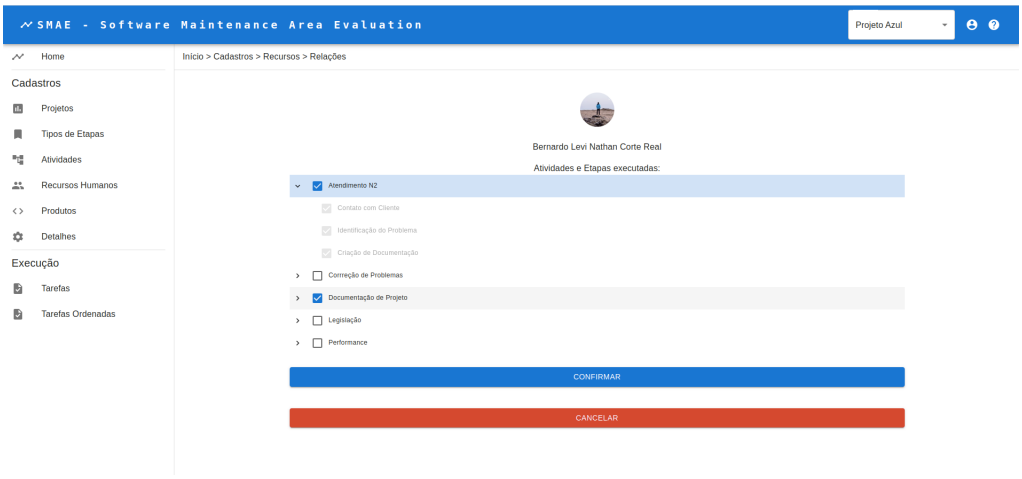
Lista de Recursos



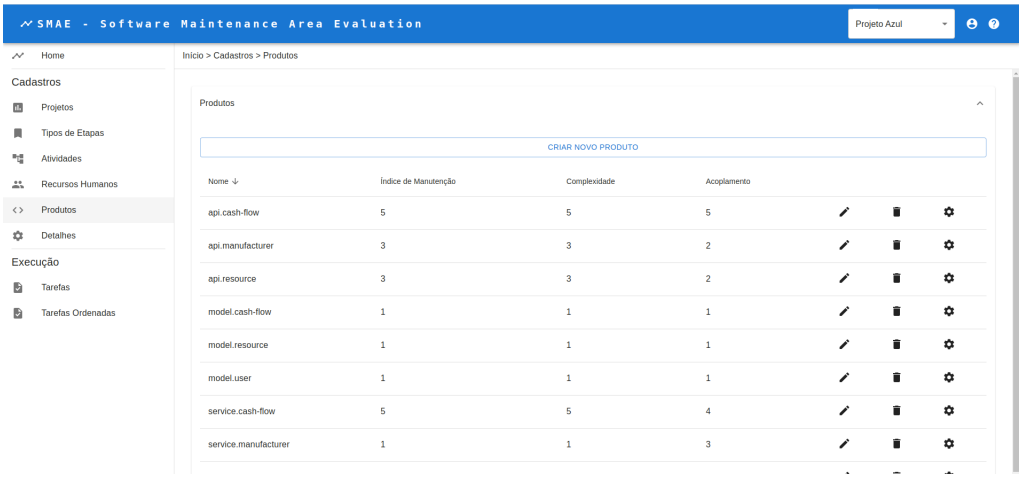
Criação de Recurso



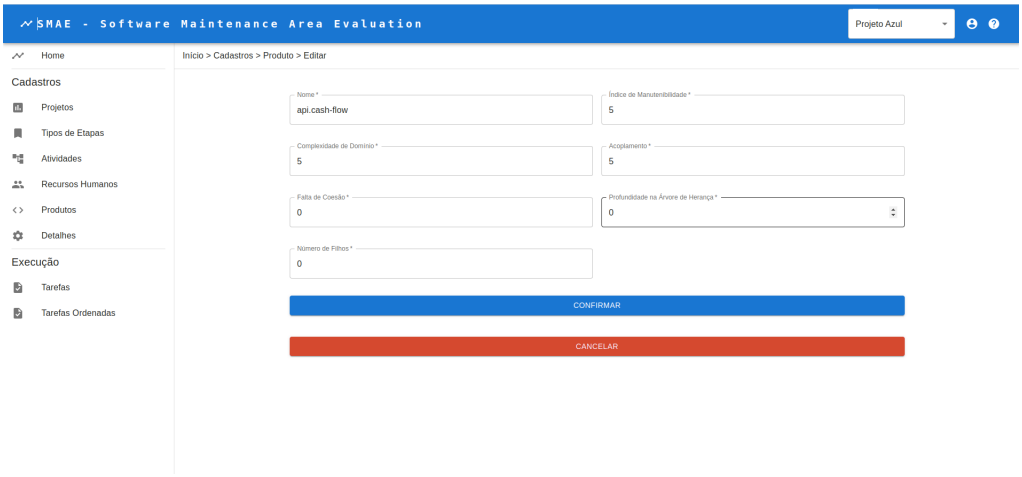
Atribuir Atividade e Etapa para Recurso



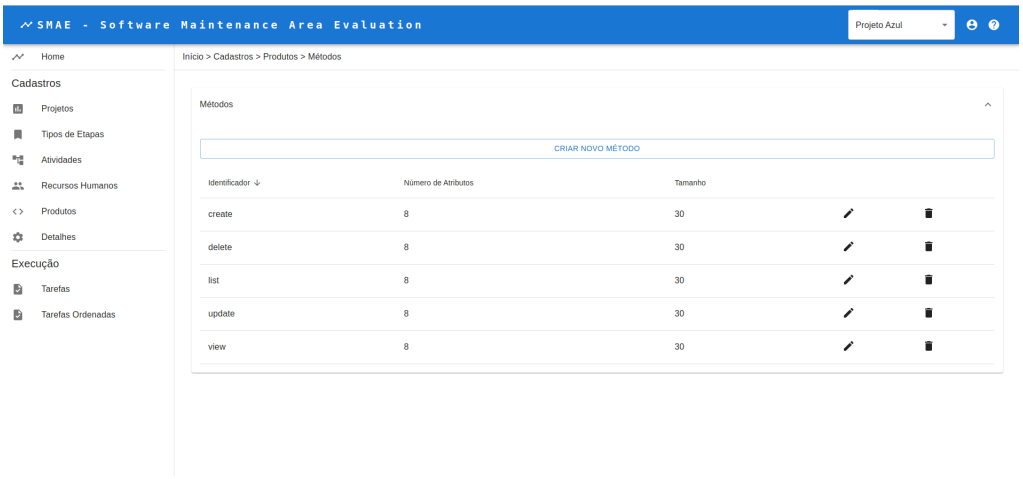
Lista de Produtos



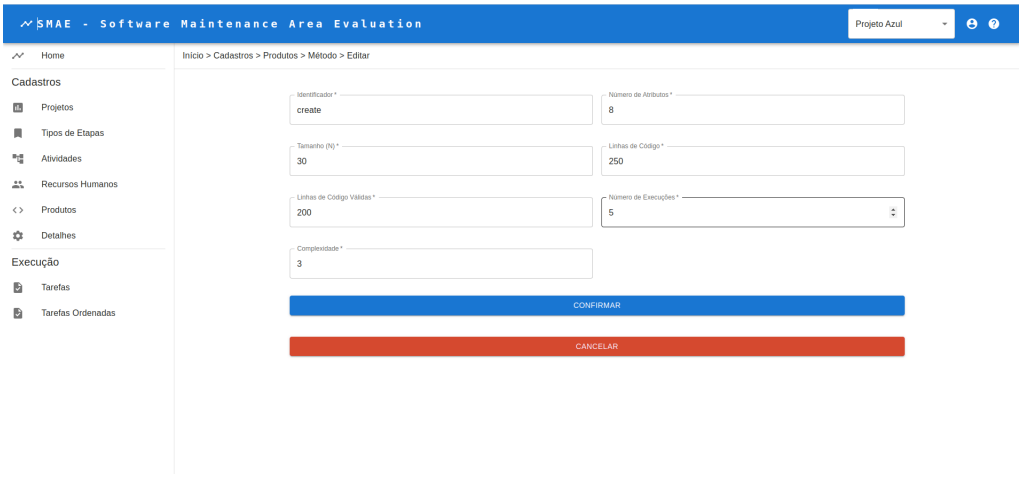
Criação de Produtos



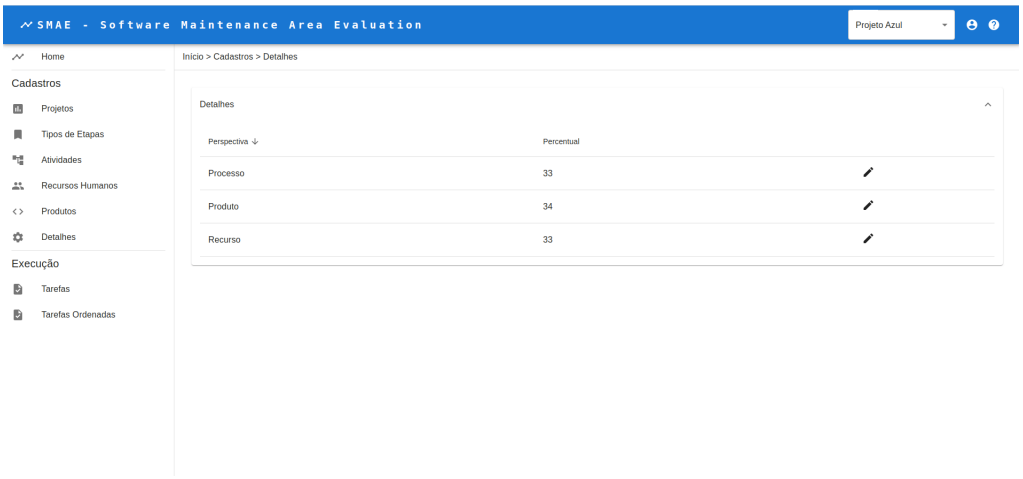
Lista de Métodos



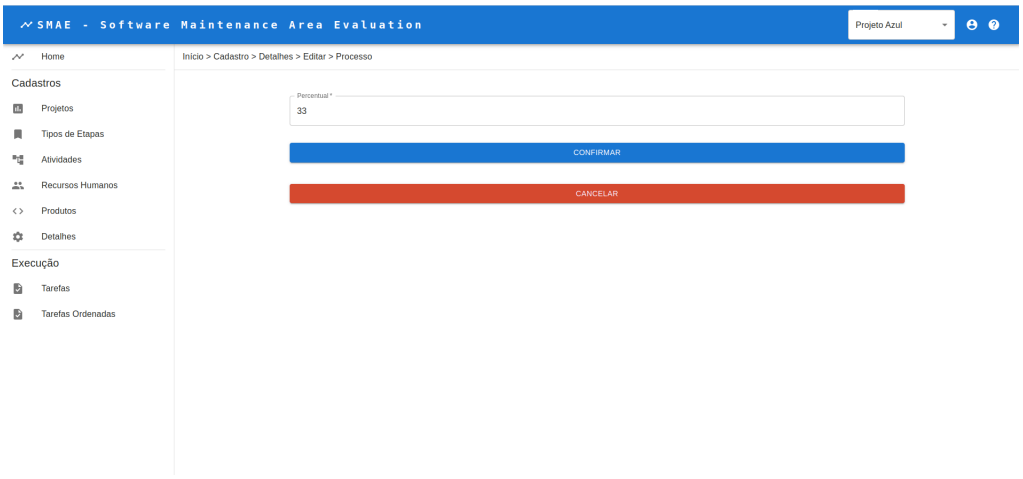
Criação de Método



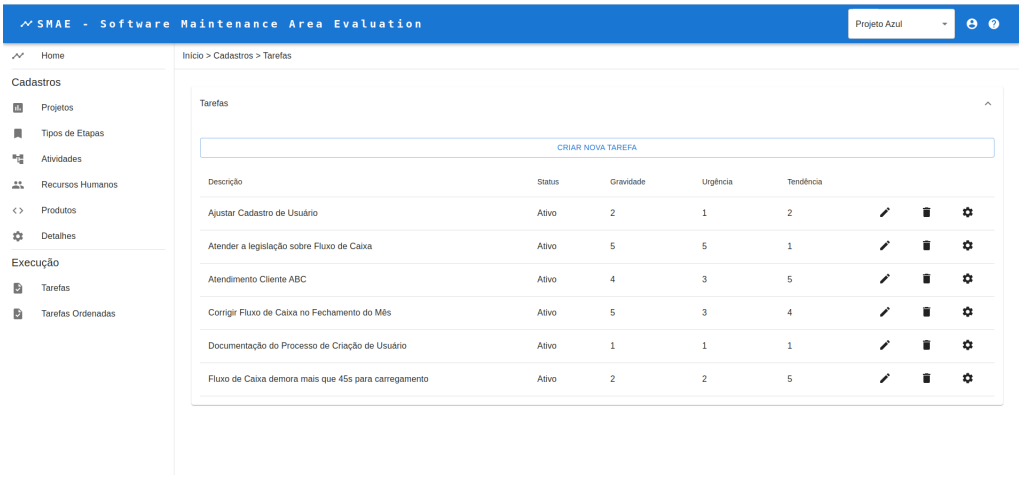
Lista de Detalhes



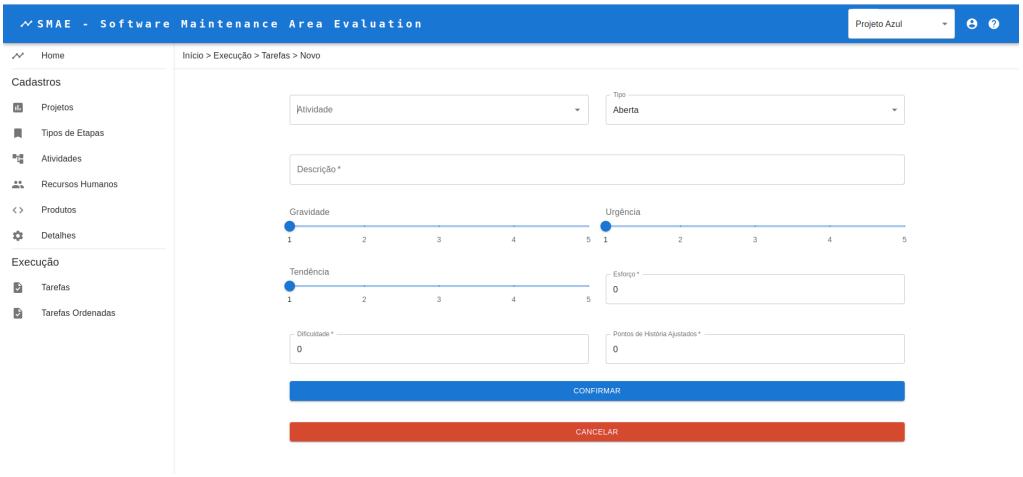
Edição de Detalhes



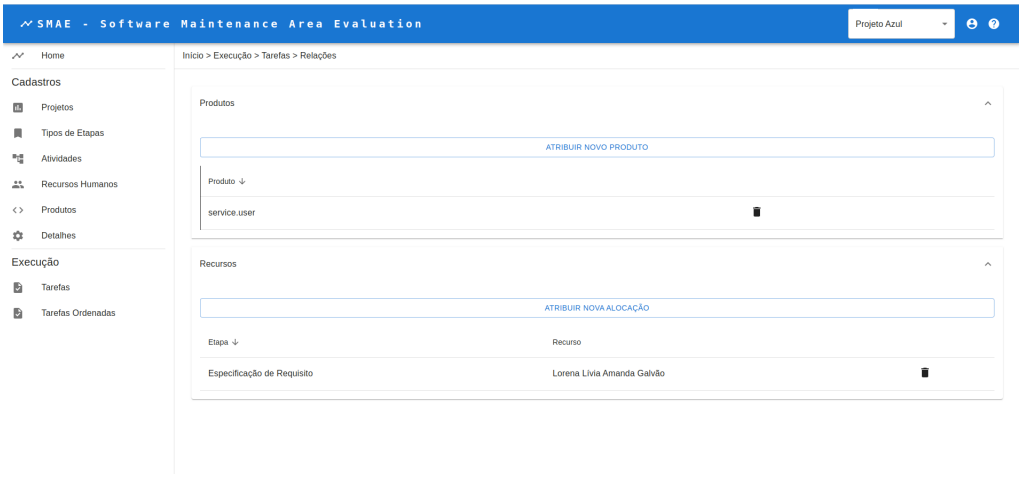
Lista de Tarefas



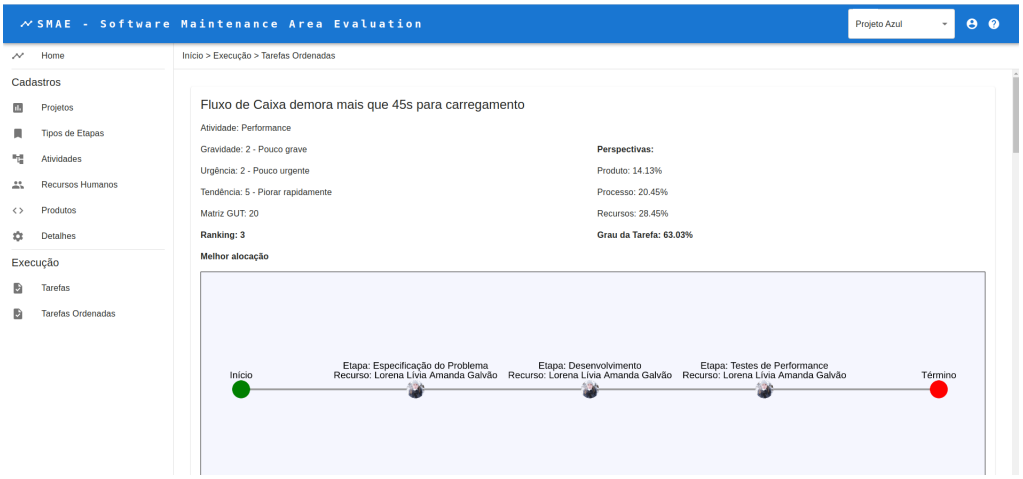
Criação de Tarefa



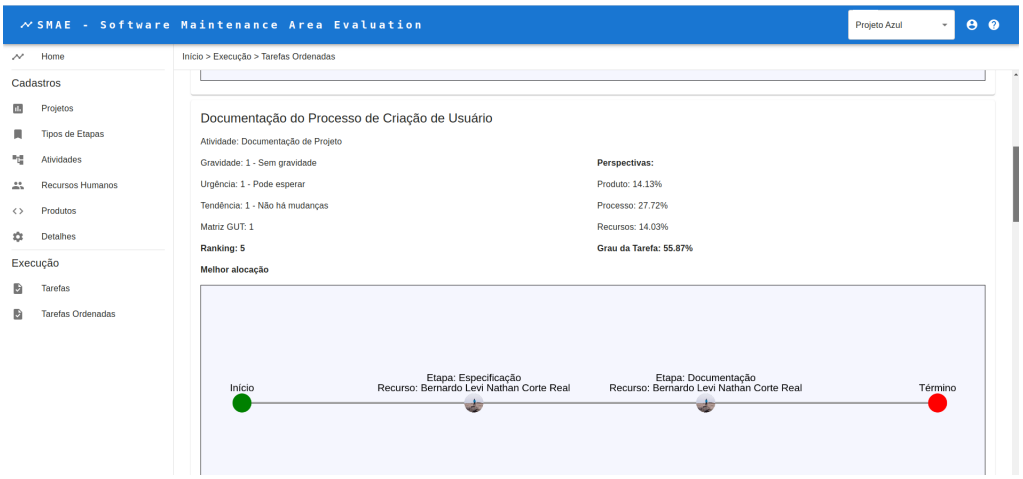
Relação de Tarefa com Produto e Recurso



Ordenação de Tarefa 1



Ordenação de Tarefa 2



ÍNDICE