Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Modelo de Apoio a Decisão para Auditoria de Código Fonte: Um Estudo de Caso

Diego Costa Sombra

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Edison Ishikawa

Brasília
2024

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

CS693m

Costa Sombra, Diego
Modelo de Apoio à Decisão para Auditoria de Código Fonte: Um Estudo de Caso / Diego Costa Sombra; orientador Edison Ishikawa. -- Brasília, 2024.
111 p.

Dissertação(Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2024.

1. Auditoria de Código Fonte. 2. Métodos, Técnicas e Ferramentas de Auditoria de Código Fonte. 3. Modelo de Apoio à Decisão por Multicritério. 4. AHP-FUZZY. 5. Segurança da Informação. I. Ishikawa, Edison, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Modelo de Apoio a Decisão para Auditoria de Código Fonte: Um Estudo de Caso

Diego Costa Sombra

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Edison Ishikawa (Orientador)
Universidade de Brasília

Prof. Dr. João Carlos Felix Souza
Universidade de Brasília

Prof. Dr. Gislane Pereira Santana
Universidade Católica de Brasília

Prof. Dr. Gladston Luiz da Silva
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 10 de maio de 2024

Dedicatória

Dedico este trabalho primeiramente a Deus, cuja sua orientação e força tornaram possível alcançar este objetivo. À minha amada esposa, Isabel Sombra, expresso profunda gratidão pela constante presença e apoio, por estar ao meu lado em todos os momentos, mesmo nos períodos em que estive ausente. Agradeço também às minhas filhas pela paciência e compreensão, mostrando-se sempre solidárias diante da importância deste projeto para o papai delas. Meu agradecimento se estende ao meu pai, um dos maiores incentivadores, à minha mãe, que sempre foi uma guerreira, e aos meus irmãos.

Agradecimentos

Sou extremamente grato aos colegas que contribuíram, direta ou indiretamente, com muita paciência, amor, energia e coragem para o desenvolvimento desta dissertação.

Agradeço profundamente ao meu orientador, Prof. Dr. Edson Ishikawa, por acreditar em mim, no projeto e por orquestrar com sabedoria e zelo cada fase do desenvolvimento do estudo.

Dedico aos membros da banca: Prof. Dr. João Carlos Feliz Souza e Prof. Dr. Gislane Pereira Santana, pela disponibilidade, senso crítico aguçado e ponderação na avaliação do projeto.

Expresso minha gratidão ao CEO Ricardo Caldas, da empresa Ativu Tecnologia, por compreender o impacto positivo do projeto e por conceder autorização para sua realização, mesmo diante de outros projetos no portfólio. Dedico também à dedicada equipe de desenvolvimento da Ativu Tecnologia pelo seu comprometimento no projeto.

Aos colegas, Prof. doutores e coordenadores do PPCA/UnB, agradeço pela parceria e pela rica troca de experiências ao longo deste curso de mestrado.

Sem dúvida, posso afirmar que meu crescimento pessoal e profissional após esta experiência é notável. Hoje, sou uma pessoa mais equilibrada, crítica e resiliente.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Com a crescente demanda no mercado de desenvolvimento de software e prazos cada vez mais apertados, impulsionados pelas metodologias ágeis, os riscos associados a esses softwares têm aumentado nos últimos anos. Os cibercriminosos aproveitam-se de falhas de segurança, qualidade e conformidade para perpetrar crimes cibernéticos contra empresas, resultando em perdas financeiras e danos à reputação das organizações. Assim, é fundamental que as empresas envolvidas no desenvolvimento e fornecimento desses softwares possam compreender como identificar e priorizar as questões que exigem tratamento imediato.

Na literatura, observou-se que os estudos existentes abordam os riscos de software de maneira isolada e não oferecem uma visão consolidada desses riscos, contendo um modelo que auxilie na tomada de decisão para priorizar qual risco e qual parte do software precisam ser tratados prioritariamente.

Em resposta a esse desafio, este estudo teve como objetivo compreender os riscos envolvidos e explorar os métodos, técnicas e ferramentas disponíveis para a validação desses riscos em um software de mercado. Após a identificação dos riscos, foram aplicados os métodos, técnicas e ferramentas no software que validou a presença desses riscos. Ao confirmar a existência dos riscos, utilizou-se o método multicritério de apoio à decisão FAHP para auxiliar na classificação do risco, determinando qual parte do software e qual risco deveria ser priorizado primeiro.

Os resultados indicaram que, dentre os nove módulos do software, o módulo **Web** com **34,57%**, em conjunto com o risco de **Vulnerabilidade** com **50,35%**, precisa ser priorizado. Este modelo de apoio à decisão surge como uma contribuição para as tomadas de decisão, especialmente na área da engenharia de software.

Palavras-chave: Auditoria de Código Fonte, Programa, AHP-FUZZY, Segurança da Informação

Abstract

With the growing demand in the software development market and increasingly tight deadlines driven by agile methodologies, the associated risks to these software have increased in recent years. Cybercriminals exploit security, quality, and compliance vulnerabilities to commit cybercrimes against companies, resulting in financial losses and damage to the organizations' reputation. Therefore, it is essential for companies involved in the development and supply of such software to understand how to identify and prioritize issues that require immediate attention.

In the literature, it was observed that existing studies address software risks in isolation and do not provide a consolidated view of these risks, lacking a model to assist in decision-making to prioritize which risk and which part of the software need to be addressed urgently.

In response to this challenge, this study aimed to understand the risks involved and explore the methods, techniques, and tools available for validating these risks in market software. After identifying the risks, methods, techniques, and tools were applied to the software, validating the presence of these risks. Upon confirming the existence of the risks, the FAHP multicriteria decision support method was used to assist in risk classification, determining which part of the software and which risk should be prioritized first.

The results indicated that, among the nine software modules, the **Web** module with **34.57%**, combined with the **Vulnerability** risk with **50.35%**, needs to be prioritized. This decision support model emerges as a contribution to decision-making, especially in the field of software engineering.

Keywords: Source Code Audit, Software, AHP-FUZZY, Information Security

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AHP Analytic-Hierarchy Process.

ANP Analytic Network Process.

BPM Bayesian Ponderation Method.

BSD Berkeley Software Distribution.

CR Critério de Risco.

ELECTRE ELimination Et Choix Traduisant la REalité.

FAHP Fuzzy Analytical Hierarchy Process.

GPL General Public License.

ICR índice de Consistência Relativa.

LGPD Lei Geral de Proteção de Dados.

LGPL Lesser General Public License.

MAUT Multi-Attribute Utility Theory.

MCDA Multicriteria Decision Analysis.

MCDM Multi-Criteria Decision Making.

MIT Massachusetts Institute of Technology.

MS Módulo Software.

OWASP Open Web Application Security Project.

PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses.

PROMETHEE Preference Ranking Organization METHod for Enrichment of Evaluations.

RSL Revisão Sistemática da Literatura.

TOPSIS Technique for Order Preference by Similarity to Ideal Solution.

Capítulo 1

Introdução

1.1 Contextualização

Recentemente, houve um aumento significativo na adoção de software em todo o mundo, impulsionado pela digitalização de processos e pela dependência crescente da tecnologia em atividades pessoais e profissionais [1]. A pandemia de COVID-19 também acelerou essa tendência, especialmente nas áreas de educação, trabalho remoto e telemedicina. O mercado global de software corporativo está em crescimento constante, com projeções indicando um aumento anual composto de 6,55% para atingir 102,99 bilhões de dólares até 2030 [1], [2]. Além disso, o mercado de software foi avaliado em 583,47 bilhões de dólares em 2022 e deve crescer a uma taxa composta anual de 11,5% de 2023 a 2030, impulsionado pela adoção de tecnologias emergentes como inteligência artificial e análise de dados [2]. Essa expansão não se limita apenas ao setor corporativo, pois a adoção de software também está aumentando entre os usuários domésticos e individuais [3].

Com o crescimento constante do mercado de software, a metodologia ágil tem sido amplamente aceita para acelerar o desenvolvimento e atender às demandas do mercado. No entanto, é preocupante que os processos de segurança, como a auditoria e revisão do código fonte, não estejam recebendo a devida prioridade. As empresas que fornecem produtos de software precisam se preocupar não apenas com a velocidade de entrega, mas também com a qualidade e segurança desses produtos [4].

A segurança da informação é um tema crítico e cada vez mais relevante no mundo atual, onde a quantidade de dados processados e armazenados nos sistemas de informação é cada vez maior, tornando-se alvo constante de ameaças e ataques cibernéticos. Nesse contexto, a auditoria de código fonte surge como uma prática essencial, envolvendo uma análise minuciosa do código fonte do software em busca de vulnerabilidades e brechas de segurança que podem ser exploradas por hackers. A auditoria de código fonte desempenha um papel fundamental na identificação de possíveis falhas de vulnerabilidades, permitindo

que sejam corrigidas antes de serem exploradas. Além disso, ela também auxilia na identificação de conformidade de licenciamento de código fonte aberto, práticas de plagio ou similaridade no código fonte, *bugs* potenciais e *code smells* que comprometem a qualidade e a segurança do software. Essas práticas evitam os riscos e aumentam a segurança dos sistemas, garantindo que os softwares fiquem protegidos e contra ameaças.

Diante deste cenário, este estudo busca compreender os riscos envolvidos e explorar os métodos, técnicas e ferramentas disponíveis para a validação desses riscos em um software de mercado. Após a identificação dos riscos, será aplicado os métodos, técnicas e ferramentas no software com o objetivo de validar a presença desses riscos. Ao confirmar a existência dos riscos, será utilizado um método de multicritério de apoio à decisão para auxiliar na classificação do risco, determinando qual parte do software e qual risco deve ser priorizado. Com este modelo de apoio à decisão, espera-se contribuir para que as áreas de desenvolvimento possam atuar com agilidade na hora de decidir o que deve ser priorizado, ajudando a manter a conformidade, qualidade e segurança de seus softwares.

1.1.1 Escopo

O projeto abordará especificamente a auditoria de código fonte, com o objetivo de identificar na literatura técnicas, métodos e ferramentas, visando obter uma análise abrangente e precisa.

Será realizado um estudo de caso em um software de mercado, no qual as técnicas, métodos e ferramentas selecionadas serão aplicadas para identificar e avaliar os critérios de risco mencionados. Serão coletados dados relevantes e analisados individualmente para compreender a natureza e a magnitude dos problemas encontrados.

Além disso, está previsto o desenvolvimento de um modelo de apoio à tomada de decisão com o propósito de orientar os gestores na formulação de uma estratégia eficaz. Este modelo disponibilizará informações fundamentadas e de fácil compreensão, que servirão de suporte para a priorização, alocação de recursos e delineamento das estratégias necessárias para a solução dos problemas identificados em auditorias de código fonte.

1.2 Definição do Problema

O processo de auditoria de código fonte é amplamente reconhecido e incorporado às estratégias das empresas. No entanto, com a crescente demanda no mercado de software e a pressão constante para entregas rápidas, observa-se uma diminuição na ênfase dada pelas equipes de desenvolvimento a esse processo crítico. Diante desse cenário, torna-se premente a necessidade de reforçar a importância desta atividade e desenvolver mecanismos automatizados que possam auxiliar as equipes de desenvolvimento a manterem o

alto desempenho em suas atividades, ao mesmo tempo em que garantem a conformidade, qualidade e segurança dos softwares.

Na literatura existente, foi encontrada diversas abordagens, métodos, técnicas e ferramentas para apoiar a auditoria de código fonte. No entanto, a maioria dessas abordagens concentra-se na identificação de problemas, carecendo de orientações detalhadas sobre como lidar com esses problemas, especialmente em cenários complexos com múltiplas alternativas. Além disso, os critérios de risco geralmente são tratados de forma isolada em estudos específicos, sem a apresentação de um modelo de tomada de decisão que possa auxiliar na priorização desses critérios em um contexto amplo de auditoria de código fonte.

Consequentemente, foi identificado uma lacuna de pesquisa, que se manifesta na ausência de modelos abrangentes que orientem a tomada de decisão no processo de avaliação e tratamento da auditoria de código fonte. Essa lacuna compromete a capacidade de melhorar o processo de revisão de software e garantir a conformidade, qualidade e segurança. Portanto, a criação de um modelo automatizado capaz de identificar, analisar e avaliar os critérios de risco em software, bem como de proporcionar uma estrutura para a priorização e tratamento desses critérios, emerge como uma necessidade fundamental para organizações dos setores público e privado que enfrentam desafios semelhantes.

1.3 Pergunta de Pesquisa

Com o objetivo de estruturar o problema de pesquisa considerando os recursos e limitação de tempo para a meta-análise, foi utilizado a estratégia PICOC [5] (acrônimo para Problema, Interesse, Controles, Resultados/Outcomes e Contexto). Seguindo esse protocolo como guia, formularam-se as seguintes perguntas de pesquisa, com o propósito de orientar a revisão sistemática da literatura (RSL):

- A) (PP_1): Quais critérios de risco são relevantes para serem avaliados em uma auditoria de código-fonte de software?
- B) (PP_2): Através da aplicação de ferramentas adequadas e métodos apropriados, é possível extrair informações relevantes e validar os critérios de risco em um software de mercado?
- C) (PP_3): Como desenvolver um modelo de tomada de decisão para ordenar as alternativas identificadas e estabelecer uma priorização eficiente considerando a complexidade de cada alternativa?

1.4 Objetivos

A auditoria de código fonte desempenha um papel fundamental na garantia da conformidade, qualidade e segurança de software. O objetivo geral deste trabalho é identificar e validar critérios de risco em software de mercado. Além disso, busca-se o desenvolvimento de um modelo que possa apoiar gestores na tomada de decisões, especialmente na priorização dos riscos que possuem maior impacto nas operações de negócio.

Para atingir esse objetivo, o trabalho buscará identificar os critérios de risco que estão envolvidos em uma auditoria de código fonte, irá examinar as técnicas e ferramentas de auditoria de código fonte para extrair informações relevantes, validar os critérios de risco em um software de mercado e construir um modelo de tomada de decisão validado em um estudo de caso para auxiliar os gerentes na priorização das ações a serem realizadas. Com este modelo de apoio à decisão, espera-se contribuir para que as áreas de desenvolvimento possam atuar com agilidade na hora de decidir o que deve ser priorizado, ajudando a manter a conformidade, qualidade e segurança de seus softwares.

A seguir, de forma sucinta, são apresentados os objetivos específicos:

- A) (OE_1): Identificar os critérios de risco, métodos e ferramentas que estão envolvidos em uma auditoria de código fonte;
- B) (OE_2): Avaliar e selecionar as ferramentas e métodos disponíveis para extrair informações relevantes e validar os critérios de risco em um software de mercado;
- C) (OE_3): Construir um modelo de tomada de decisão que auxilie na priorização das ações de tratamento, considerando a relevância diante da complexidade de cada alternativa;
- D) (OE_4): Validar o modelo proposto por meio do cálculo de consistência e análise de sensibilidade, considerando as opiniões dos especialistas da empresa.

1.5 Justificativa

Um software eficaz é aquele que atinge seu objetivo e oferece uma experiência agradável ao usuário em um ambiente seguro. No entanto, é importante garantir que os softwares atendam a todos os critérios de sucesso definidos [6] [7]. A segurança do software é

muitas vezes negligenciada durante o processo de desenvolvimento, levando a problemas de segurança no final do projeto [8] [9].

A ausência de medidas sistemáticas, como revisões e procedimentos, pode ocasionar em softwares que são percebidos como uma ameaça. [10]. É essencial adotar uma abordagem ou estrutura que seja utilizada desde o início do projeto para garantir medidas de segurança efetivas [8] [11]. Ao incorporar o pensamento voltado para a segurança em todo o processo de desenvolvimento, é possível descobrir problemas de segurança mais cedo e evitar erros críticos em projeto de software [12]. Isso reduz o risco de perder requisitos de segurança importantes e torna a correção de problemas mais fácil e econômica.

Diversos incidentes de violação de dados, como o ocorrido no Citibank [13], em que mais de 360 mil números de cartão de crédito foram roubados e no hack DAO [14], que um hacker desconhecido explorou várias vulnerabilidades no código DAO e ganhou controle de cerca de 40 milhões de dólares na época, destacam a importância da segurança de software. As vulnerabilidades em bibliotecas e ambientes de execução, bem como ataques como o ransomware WannaCry [15], que se espalhou por pelo menos 150 países, afetando mais de 200 mil usuários e causou uma perda de 8 bilhões de dólares, representam ameaças para empresas, governos e consumidores. De acordo com um estudo recente da empresa IBM [16], o custo médio de dados violados aumentou em 12% nos últimos 5 anos, chegando a 3,92 milhões de dólares em 2019. Isso reflete o crescente impacto financeiro e os riscos associados à proteção de dados e ressalta a importância de medidas efetivas de segurança para proteger informações sensíveis.

Outro aspecto preocupante é o uso indevido de bibliotecas de terceiros e trechos de código sem autorização, o que pode levar a violação de direitos autorais com penalidades graves para empresas que praticam este ato, assim, a clonagem de fragmentos de código requer a permissão explícita do proprietário dos direitos autorais dos fragmentos [17].

Além disso, uma outra preocupação recente para empresas que fabricam software é a aprovação da Lei Geral de Proteção de Dados (LGPD) no Brasil que traz a necessidade de aperfeiçoamento dos aplicativos que tratam dados pessoais dos cidadãos, sob pena de multas substanciais. A LGPD considera como pessoal quaisquer dados que direta ou indiretamente levem à identificação de um usuário. Negligenciar as exigências da LGPD pode significar multas de até 2% da receita global das empresas [1].

Assim, esta pesquisa visa explorar métodos e ferramentas para visualizar o cenário atual de segurança, qualodade e conformidade do software e criar um modelo de apoio à decisão para priorizar e organizar as ações de tratamento de riscos identificados. Com o apoio desse modelo, as organizações poderão tomar decisões mais consistentes e eficientes na proteção de seus dados e de seus clientes.

[1] `http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm`

Portanto, este estudo se justifica diante da necessidade de proteger informações sensíveis, garantir a conformidade, qualidade e segurança. O resultado dessa pesquisa pretende trazer benefícios para as organizações que enfrentam desafios semelhantes na gestão de seus softwares.

1.6 Metodologia

A metodologia adotada neste estudo representa a implementação de diversas técnicas e ferramentas que visam facilitar a realização das tarefas propostas. Enquanto o modelo estabelece as diretrizes do que precisa ser feito, a metodologia vai definir a forma de executar essas atividades [18]. Nesta seção, é apresentado a estrutura metodológica utilizada para atingir os objetivos alcançados. Com base na abordagem qualiquantitativa e seguindo as etapas delineadas por Marconi e Lakatos [19] e Prodanov e De Freitas [20], este estudo é caracterizado como **descritivo** e **exploratório**, com o propósito de identificar e validar os critérios de risco identificados na literatura e construir um modelo de apoio à tomada de decisão para auxiliar os gestores, proporcionando a priorização e tratamento adequados dos critérios de risco que estão envolvidos em uma auditoria de código fonte.

1.6.1 Classificação e Fases da Pesquisa

O planejamento da pesquisa estabelece uma sequência de etapas que deve ser seguida dentro de um cronograma definido, visando resolver um problema específico [21]. Nesse sentido, a metodologia deste estudo concentra-se em macroprocessos inter-relacionados, conforme ilustrado na Figura 1.1.

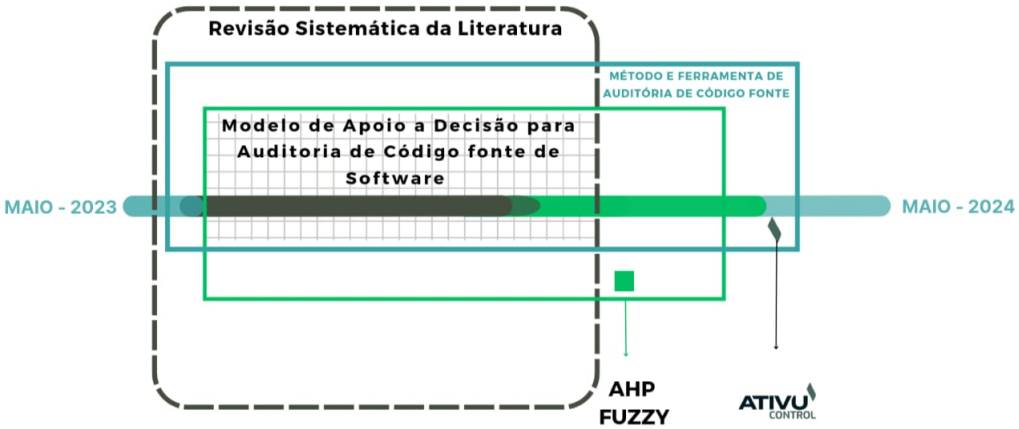


Figura 1.1: Relacionamento Metodológico - Principais Áreas da Pesquisa
Fonte: Adaptado de Triperina.[22].

Quanto à sua natureza, esta pesquisa é classificada como **aplicada**, pois busca oferecer conhecimento direcionado para a efetiva solução de um problema em um estudo de caso. O objetivo é contribuir, na prática, para melhorar o processo de auditoria do código fonte.

Esta pesquisa é classificada como **exploratória**, uma vez que busca adquirir e analisar conhecimentos práticos sobre a produção de um modelo de apoio à decisão para priorizar critérios de risco na auditoria de código fonte, preenchendo uma lacuna na literatura. Além disso, ela também é classificada como **descritiva**, pois considera uma revisão sistemática da literatura (RSL) realizada para identificar os principais critérios de risco, ferramentas e métodos de auditoria de código fonte, bem como o uso do (MCDA) em um estudo de caso para construir um modelo que auxilie nas decisões dos especialistas e gestores responsáveis pelo desenvolvimento de software.

1.6.2 Plano Metodológico

Com base no exposto, a condução deste trabalho é estruturada em quatro fases principais: Planejamento, Execução, Estudo de Caso e Constatação. Essas fases seguem a sequência de procedimentos e métodos delineados no Plano Metodológico, que podem ser visualizados na Figura 1.2. Cada uma dessas fases, assim como suas etapas correspondentes, foram adaptadas ao contexto específico deste estudo, tomando como referência os eixos/fases propostas por Gerhardt e Silveira [23] e Prodanov e De Freitas [20], respectivamente ilustrados a seguir:

A metodologia de pesquisa adotada neste estudo segue três eixos/fases fundamentais, conforme escrito por Gerhardt e Silveira [23] e Prodanov e De Freitas [20]:

Ruptura/Decisória: Este eixo busca questionar e superar conceitos baseados apenas em senso comum e evidências não fundamentadas em dados empíricos, e nesta etapa, ocorre a definição clara do tema e do problema de pesquisa a ser investigado.

Construção/Redacional: Aqui, é desenvolvido um sistema conceitual, formal e organizado, que pode ser compreendido e replicado no plano de pesquisa em questão, e a metodologia é posta em prática, seguindo o plano estabelecido, a fim de coletar dados e informações relevantes para a análise.

Constatação/Construtiva: Nesta etapa, as proposições elaboradas são desenvolvidas, fundamentadas nos fatos e comprovadamente verificadas durante o estudo, e na última fase, são apresentados os resultados obtidos, assim como as considerações finais do estudo.

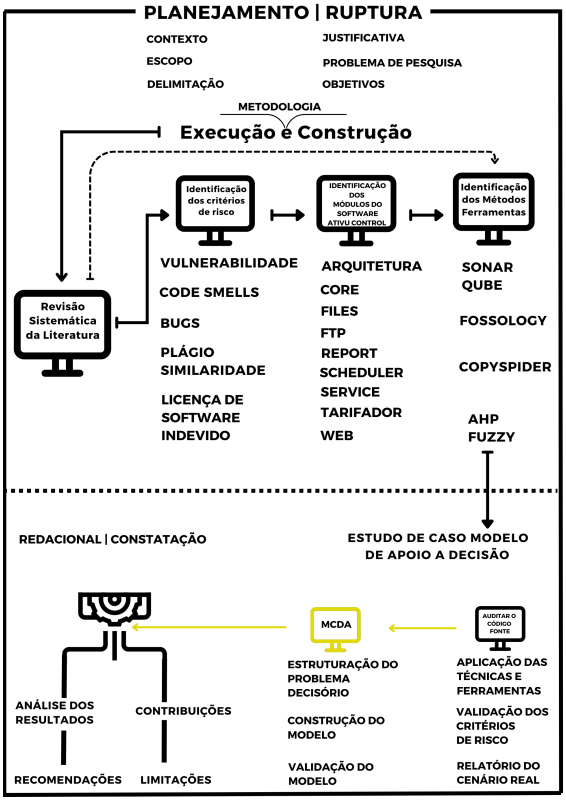


Figura 1.2: Fases e Etapas da Pesquisa.
Fonte: Elaboração própria.

Após concluída a Fase Planejamento/Ruptura, as Fases Execução, Estudo de Caso e Constatação se elevam em etapas fundamentais, que constituem os instrumentos necessários para atingir os objetivos desta pesquisa, descritos a seguir:

- Foi realizada uma Revisão Sistemática da Literatura (RSL), guiada pelo protocolo (PRISMA), que orientou a pesquisa bibliográfica na **identificação dos critérios de risco**, bem como dos **métodos e ferramentas utilizadas em auditoria de código fonte**, conforme específicos nos trabalhos relacionados;

- A **aplicação** dos métodos e ferramentas de auditoria de código fonte no software não apenas validou os critérios de risco identificados na Revisão Sistemática da literatura (RSL), mas também apontou uma visão real dos problemas encontrados;

- A abordagem de Multicriteria Decision Analysis (MCDA) foi utilizada para a **Estruturação do Problema Decisório e construção do modelo de apoio à decisão** padrão neste trabalho;

- O cálculo de consistência, a Análise de Sensibilidade e a Entrevista não estruturada

foram realizadas com representantes decisores no estudo de caso para validar o escopo da pesquisa e os critérios identificados, com o objetivo de obter a **Validação do Modelo** padrão.

1.7 Contribuição Esperada da Pesquisa

Espera-se que esta pesquisa contribua para a área de auditoria de código fonte por meio da revisão das técnicas existentes na literatura e da identificação de critérios relevantes para a auditoria de código fonte. Além disso, espera-se que a avaliação das técnicas de auditoria com uma ferramenta automatizada e o estudo de caso em um ambiente real possam fornecer informações para a elaboração de um modelo de auditoria de código fonte baseado nas técnicas escolhidas. A contribuição final desta pesquisa será a apresentação de um modelo de auditoria de código fonte. Este modelo irá auxiliar as equipes de desenvolvimento na revisão de seus códigos, assegurando simultaneamente a conformidade, qualidade e segurança, sem que isso comprometa o desempenho no processo de desenvolvimento.

1.8 Estrutura do Documento

Esta dissertação está organizada em seis capítulos, que fornecem uma estrutura para a compreensão e análise do tema proposto. A seguir, apresenta-se uma breve descrição de cada capítulo:

No **Capítulo 1**, apresenta uma contextualização sobre o tema, justifica o estudo, expõe o problema de pesquisa e os objetivos do trabalho. Além disso, são delineados o escopo e a metodologia utilizada.

No **Capítulo 2**, são abordados os principais conceitos, teorias e modelos relacionados ao tema da dissertação. São explorados os fundamentos da auditoria de código fonte, bem como as técnicas, ferramentas e critérios de análise utilizados nesse processo.

No **Capítulo 3**, para contextualizar a proposta desta dissertação, serão apresentados trabalhos semelhantes que influenciaram o desenvolvimento desta pesquisa. Serão exploradas referências que abordam temáticas semelhantes ou que serviram como base para fundamentar o presente estudo.

No **Capítulo 4**, é realizado à validação dos critérios de risco identificados na Revisão Sistemática da Literatura (RSL) por meio de sua aplicação em um software de mercado. Além disso, é construído um modelo de apoio à decisão para priorização do tratamento dos riscos, utilizando o método FAHP. Também é realizada a validação do modelo, verificando sua consistência, e robustez das decisões adotadas através da análises de sensibilidade.

Enquanto no **Capítulo 5**, é apresentado uma discussão detalhada sobre os resultados obtidos e as perspectivas deste trabalho em relação às suas contribuições tecnológicas e institucionais. Além disso, é fornecido um Modelo de Apoio a decisão para nortear os especialistas envolvidos no processo de auditoria de código fonte, bem como para outras pessoas envolvidas.

Por fim, no **Capítulo 6**, é apresentado as conclusão a partir deste estudo, destacando os principais resultados obtidos, contribuições identificadas ao longo da pesquisa. Além disso, é compartilhado as experiências práticas e teóricas adquiridas durante o trabalho, proporcionando uma visão do percurso percorrido, apontando as abordagens possíveis para pesquisas futuras, a fim de incentivar o aprofundamento do conhecimento e o desenvolvimento contínuo na área de pesquisa.

Capítulo 2

Fundamentação Teórica

Neste capítulo, é apresentada a fundamentação teórica com a finalidade de fornecer as definições e conceitos relacionados à auditoria de código fonte, apresentando informações sobre os respectivos critérios de risco que compõem o conjunto de princípios desse ambiente, juntamente com os métodos e ferramentas envolvidas no processo. O objetivo é responder às seguintes perguntas ao final deste capítulo: O que é auditoria de código fonte? Quais são os critérios de risco envolvidos? Quais são os métodos e ferramentas utilizadas em una auditoria de código fonte? Existe algum método que ajude na tomada de decisão?

2.1 Auditoria de Código Fonte

Uma revisão ou auditoria de código fonte tem como objetivo identificar problemas de segurança e violação das especificações do programa. A presença de *bugs*, códigos mal escritos como *code smell* ou vulnerabilidades é comum em software e pode resultar em diversos problemas relacionados à segurança e qualidade, como impasses, vazamento de informações, lentidão e travamento do sistema [24]. Além disso, a auditoria permite verificar também a existência de plágio ou semelhança de código fonte e uso indevido de licenças de empresas terceiras, ou que foi definido como cópia não reconhecida de documentos ou programas [25].

Com o rápido desenvolvimento de ideias de código aberto, muitos documentos e códigos fonte estão disponíveis na internet e são facilmente acessíveis [26]. Portanto, é necessário auditar essas variáveis para evitar problemas para a empresa. Dentre as boas práticas, todo código, biblioteca ou funcionalidade de fontes externas, como Stack Overflow ou GitHub, deve ser auditado antes de ser incorporado como recurso no sistema.

2.1.1 Vulnerabilidade

A vulnerabilidade de código fonte é uma fraqueza ou falha em um software, geralmente decorrente de erros de programação, que pode ser explorada por cibercriminosos para comprometer a segurança do sistema ou obter acesso não autorizado a informações restritas. Essas vulnerabilidades podem permitir que cibercriminosos executem códigos maliciosos, acessem dados sensíveis, causem falhas no sistema ou explorem outras brechas de segurança [27].

O número de software web contendo vulnerabilidades de alta gravidade aumentou em 2020 e 2021 em comparação com 2019, conforme ilustrado na figura 2.1.

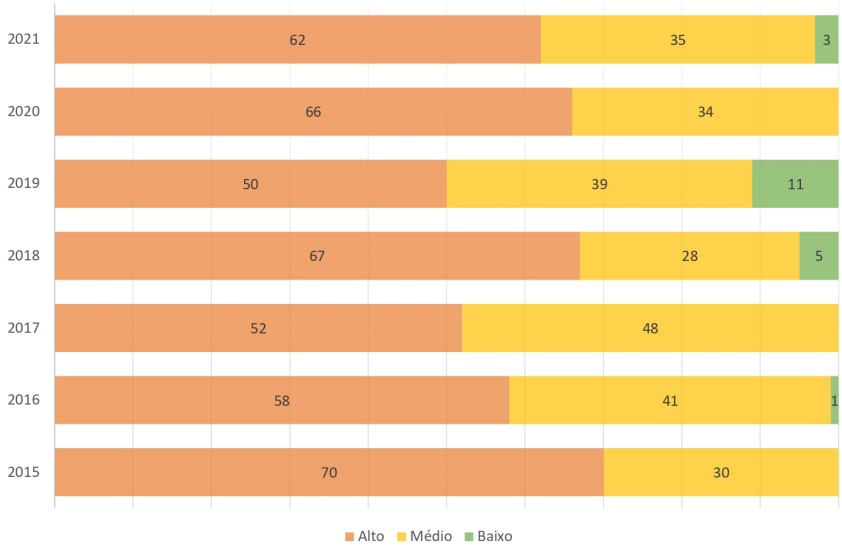


Figura 2.1: Software da Web - Comparativo Anual de Vulneráveis por Gravidade
Fonte: Adaptado de Positive Technologies [27].

As vulnerabilidades de código fonte podem assumir diversas formas, como estouro de buffer, injeção de código, falta de validação de entradas, autenticação fraca, entre outras. Elas podem ocorrer durante o processo de desenvolvimento do software devido a erros humanos, falta de conhecimento em segurança ou pressão para entregar o produto rapidamente, ou podem surgir devido a mudanças no ambiente de ameaças e novas técnicas de ataque [28]. A OWASP apresenta o top10 dos principais risco de segurança que podem ocorrer em um Software Web, que estão ilustradas na figura 2.2.

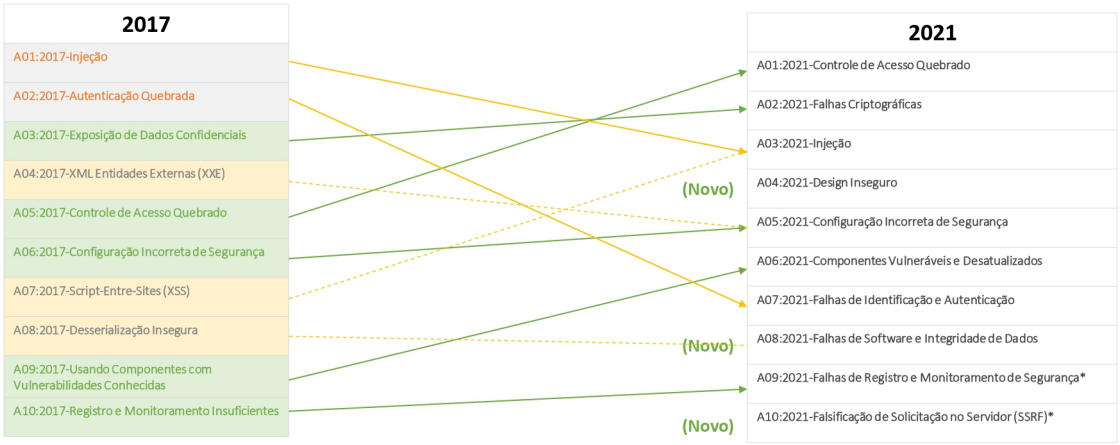


Figura 2.2: Os 10 Principais Riscos de Segurança de Aplicativos da Web
Fonte: Adaptado de OWASP [28].

É essencial identificar e corrigir essas vulnerabilidades o mais cedo possível no ciclo de desenvolvimento do software, por meio de auditorias de código, revisão de segurança e adoção de práticas de programação segura. A falta de tratamento adequado dessas vulnerabilidades pode deixar o software suscetível a ataques, comprometendo a integridade, confidencialidade e disponibilidade dos dados e sistemas envolvidos [28][27].

2.1.2 *Code Smells*

Code Smells, em português cheiros de código, referem-se a certos padrões de código ou estruturas que indicam a possibilidade de problemas no design ou na implementação de um software. São sinais de que o código pode estar mal projetado, difícil de entender, manter ou estender, e que pode conter problemas de segurança ou erros [29]. Esses cheiros não são erros de programação que causam falhas imediatas no software, mas são indícios de que o código pode ser melhorado para torná-lo mais limpo, eficiente e seguro. Eles geralmente surgem devido a decisões de design mal pensadas, práticas de programação avançadas ou falta de atenção a detalhes importantes.

Os *Code Smells* incluem duplicação de código, métodos muito longos ou com muitos parâmetros, classes com muitas responsabilidades, nomes de variáveis ou funções pouco descritivas, uso excessivo de condicionais aninhadas, entre outros.

Alguns exemplos comuns de *Code Smells* são:

- O *Blob* é um *Code Smell* que se refere a um pedaço de código grande e complexo, que geralmente possui muitas responsabilidades e funcionalidades misturadas. Um *Blob* pode ser identificado em um código fonte quando uma única classe ou método é responsável por uma grande quantidade de tarefas diferentes e não relacionadas. Isso pode tornar o código difícil de entender, manter e modificar, pois todas as alterações precisam ser feitas em um único local, afetando várias partes do sistema. A presença de um *Blob* no código pode ser um indicativo de que o design do software precisa ser melhorado e refatorado para dividir as responsabilidades em classes ou métodos menores e mais coesos [30].

- O *Feature Envy* surge quando um método de uma classe usa mais recursos de outra classe do que de sua própria classe. Em outras palavras, esse cheiro ocorre quando um método está desejando os atributos e métodos de outra classe, o que pode indicar uma má organização ou design do código. O *Feature Envy* é problemático porque viola o princípio de encapsulamento, uma vez que exibe abertamente os detalhes internos de outra classe ao método atual. Isso pode tornar o código mais difícil de manter, pois qualquer alteração na classe desejada pode ter impacto em várias partes do código [29].

- O *Parallel Inheritance* ocorre quando duas ou mais classes estão interligadas de forma paralela, ou seja, quando há uma relação de herança entre classes e essa relação é replicada em outra autoridade de classes. Esse problema pode tornar o código complexo, difícil de entender e manter, e levar a um design pouco flexível. Esse *Code Smell* é uma ocorrência comum quando há uma herança entre classes de domínios diferentes, provocada em uma estrutura paralela que não reflete corretamente as relações reais entre os objetos do sistema [29].

- O *Shotgun Surgery* refere-se a uma situação em que uma pequena alteração no código exige modificações em várias partes diferentes do sistema. Essas modificações são frequentemente dispersas por várias classes ou módulos, o que torna a manutenção e o entendimento do código mais difícil. Esse problema surge quando a lógica relacionada está espalhada em várias partes do código em vez de estar centralizada em um único local. Como resultado, ao fazer uma alteração em uma funcionalidade específica, é necessário fazer várias modificações em diferentes partes do código, o

que aumenta a chance de erros e torna a manutenção do sistema mais complexa [29].

- O *Divergent Change* é uma situação em que uma única classe é frequentemente alterada em várias partes diferentes do código para adicionar ou modificar funcionalidades não relacionadas. Isso ocorre quando uma classe está assumindo muitas responsabilidades diferentes e, como resultado, torna-se propensa a mudanças frequentes e extensas. Quando uma classe sofre Mudança Divergente, isso indica uma falta de coerência, ou seja, a classe está tratando de muitas tarefas não relacionadas, o que dificulta a clareza do código e dificulta a manutenção [29].

Dessa forma, identificar e corrigir *Code Smells* é uma prática importante no desenvolvimento de software, pois ajuda a melhorar a qualidade do código, tornando-o mais legível, reutilizável, evitando problemas de qualidade e segurança.

2.1.3 *Bugs*

Os *Bugs* no código fonte são erros ou falhas de programação que causam problemas em um software. Esses *Bugs* podem levar a problemas de funcionalidade, desempenho ou segurança no programa.[31] São resultado de erros cometidos durante o processo de desenvolvimento do software, como equívocos na lógica, uso incorreto de funções ou variáveis, erros de programação, entre outros.

Os *Bugs* podem ter diferentes níveis de gravidade, desde pequenos problemas que tiveram apenas a usabilidade do software até erros críticos que causam falhas ou quebras no sistema. Além disso, *Bugs* podem ser encontrados em todas as etapas do desenvolvimento de software, desde a fase de incentivo até a implantação e uso do programa em produção[32].

Identificar e corrigir *Bugs* é uma parte essencial do ciclo de desenvolvimento de software. Testes rigorosos, revisão de código e adoção de boas práticas de programação são algumas das abordagens utilizadas para minimizar a ocorrência de *Bugs* no código fonte. A correção oportuna de *Bugs* é crucial para garantir a qualidade, confiabilidade e segurança do software, além de evitar problemas e prejuízos aos usuários e às empresas que utilizam o programa.

2.1.4 Plágio ou Similaridade

O plágio é caracterizado como uma cópia não reconhecida de documentos ou programas [25], tornou-se uma preocupação crescente em meio ao rápido desenvolvimento de ideias

e códigos abertos disponíveis na internet [26]. Com a facilidade de acesso a uma ampla gama de documentos e códigos fonte, torna-se fundamental promover a conscientização sobre a importância do reconhecimento de autoria e garantir a integridade da propriedade intelectual no ambiente digital. O plágio ou similaridade no código fonte é quando trechos ou partes do código de um software são copiados e usados sem a recepção ou permissão do autor original. Assim como em trabalhos acadêmicos, onde o plágio envolve copiar ideias, textos ou dados sem citar a fonte, no contexto de desenvolvimento de software, o plágio ocorre quando blocos de código são reproduzidos sem autorização ou referência à fonte original [33].

Existem diferentes formas de Plágio ou Similaridade no código fonte, incluindo:

- Clonagem de código que consiste na cópia literal de trechos ou mesmo de um código inteiro de outra fonte, como um projeto open source, sem alterações ou recebido.
- Similaridade estrutural que envolve a recriação de algoritmos ou estruturas de controle de fluxo semelhantes ao código de outra fonte, mesmo que o código em si não seja uma cópia exata.

O plágio ou similaridade no código é considerado uma prática antiética e, em muitos casos, viola direitos autorais. Além disso, pode resultar em consequências legais e éticas para o autor que cometeu a infração, bem como para uma organização ou empresa que utiliza o código plagiado [33].

2.1.5 Uso de Licença Indevido

A utilização restrita de licença ou biblioteca no código fonte ocorre quando um desenvolvedor incorpora componentes de software ou bibliotecas de terceiros em seu projeto sem seguir corretamente as condições protegidas nas respectivas licenças de uso. As licenças de software são documentos legais que definem os termos e condições para o uso, distribuição e modificação do código fonte e/ou do software em questão. Existem várias licenças de software, cada uma com suas próprias regras e restrições. Algumas licenças permitem a distribuição e o uso livre do software, incluindo o uso em projetos comerciais, desde que os devidos créditos sejam atribuídos ao autor original. Outras licenças podem ter restrições mais rígidas, como exigir que o código fonte modificado seja disponibilizado publicamente [34]. Quando um desenvolvedor utiliza uma biblioteca ou componente de software de terceiros, ele precisa cumprir as condições da licença associada a essa biblioteca. Isso inclui cumprir requisitos específicos, como incluir avisos de direitos autorais, divulgar as modificações feitas no código fonte ou mencionar o autor original nos créditos do software final [35].

Alguns tipos comuns de licenças são:

- Licença MIT [1] é uma licença permissiva de software livre que permite que o software seja utilizado, modificado e redistribuído livremente, desde que a nota de direitos autorais e a licença sejam incluídas em cópias e modificações. Ela é considerada uma licença permissiva porque não exige que o código fonte seja disponibilizado publicamente [36].

- Licença BSD [2] é outra licença permissiva de software livre. Ela permite que o software seja utilizado, modificado e redistribuído livremente, desde que a nota de direitos autorais e a licença sejam incluídas em cópias e modificações. Como a Licença MIT, a Licença BSD não exige que o código fonte seja disponibilizado publicamente [36].

- Licença GPL [3] é uma licença Copyleft de software livre. Ela permite que o software seja utilizado, modificado e redistribuído livremente, desde que todas as modificações e novos trabalhos baseados no software estejam sob a mesma licença GPL. Isso significa que qualquer pessoa que distribua uma versão modificada do software deve tornar o código fonte disponível publicamente sob a mesma licença GPL [36].

- Licença LGPL [4] é uma licença Copyleft semelhante à Licença GPL, mas com algumas diferenças. Ela é projetada para ser usada com bibliotecas de software, permitindo que o software que utilize a biblioteca seja licenciado sob outras licenças. A LGPL exige que as modificações na biblioteca sejam disponibilizadas publicamente sob a mesma licença LGPL, mas permite que o software que utilize a biblioteca seja licenciado sob outras licenças [36].

Os sistemas são protegidos legalmente por quatro variáveis, marca de registro, segredos comerciais, patentes e direitos autorais. A utilização indevida de licença ou biblioteca no código fonte pode levar a consequências legais e éticas. [37], [38], [39], [40]. Por isso, é importante que os desenvolvedores estejam cientes das licenças associadas às bibliotecas que usam e seguem corretamente os termos estabelecidos para garantir o uso legal e ético do software desenvolvido.

[1] Massachusetts Institute of Technology.
[2] Berkeley Software Distribution.
[3] General Public License.
[4] Lesser General Public License.

2.2 Métodos, Técnicas e Ferramentas de Auditoria de Código Fonte

Nos últimos anos, a auditoria de código em aplicativos da Web era realizada principalmente de forma manual, o que dificultava garantir a eficiência e precisão do processo. Essa abordagem demandava mais tempo e mão de obra, especialmente em projetos de grande porte [41]. Como resposta a essa necessidade, diversos métodos, técnicas e ferramentas foram aprimoradas para auxiliar e complementar o trabalho de auditoria, tornando o processo mais ágil e efetivo [42].

2.2.1 Métodos e Técnicas

Na atualidade, os métodos de análise de código fonte em software podem ser categorizados em duas classes distintas: análise estática e análise dinâmica. Pesquisadores de diversas origens, tanto nacionais quanto internacionais, têm dedicado considerável esforço nesse campo, resultando em uma ampla gama de abordagens e estruturas para a análise de código fonte [42].

A análise estática é realizada sem a execução do código e analisa o software em um nível mais abstrato. É uma técnica que examina o código fonte ou o bytecode para encontrar possíveis erros, violações de padrões de codificação, vulnerabilidades de segurança e outros problemas que possam ser detectados sem a necessidade de executar o programa. A análise estática pode ser feita de forma manual ou automatizada por meio de ferramentas especializadas [42].

A análise dinâmica é realizada durante a execução do programa. Nessa técnica, a ferramenta analisa o comportamento do software em tempo de execução, verificando o fluxo de dados, identificando vulnerabilidades de segurança e detectando problemas de desempenho que possam ser causados pela interação com o ambiente de execução [43]. A análise dinâmica geralmente é mais complexa e pode ser mais demorada do que a análise estática.

Tanto a análise estática quanto a dinâmica podem integrar o aprendizado de máquina em seus processos, incorporando ferramentas e técnicas que utilizam algoritmos de aprendizado de máquina para identificar padrões mais complexos e contextuais. Na análise estática, essas ferramentas empregam técnicas de mineração de dados e aprendizado de máquina para identificar padrões e anomalias no código fonte, sem a necessidade de executá-lo. Por outro lado, na análise dinâmica, as ferramentas executam o código e monitoram seu comportamento, coletando informações sobre o uso de recursos, interações com o sistema operacional, entrada e saída de dados, entre outros aspectos [44].

Em resumo, a diferença principal entre análise estática e dinâmica reside no fato de que a análise estática examina o código fonte ou o *bytecode* sem executar o programa e a análise dinâmica observa o comportamento do software em execução. Ambas as abordagens, quando combinadas com aprendizado de máquina, desempenham um papel crucial na detecção de problemas no software. Uma abordagem integrada que tire proveito de ambas as técnicas pode ser mais eficaz para identificar questões que afetam a segurança e qualidade do software [45]. Contudo, dado o desafio de combinar as duas técnicas, pode ser viável selecionar a técnica mais adequada no momento e, posteriormente, aprimorar o modelo com a colaboração de especialistas [46].

2.2.2 Ferramentas

A abordagem para resolver problemas de segurança deve começar no estágio inicial do código fonte. A avaliação da segurança do código fonte é apoiada por ferramentas automatizadas de auditoria, combinadas com análise artificial. Essas ferramentas inicialmente rastreiam problemas no código e, em seguida, a análise, correção e validação dos problemas são executados por intervenção humana. À medida que as linguagens de programação evoluem, as tecnologias de análise de código fonte também se aprimoram. O cenário de segurança da informação é pontuado por várias ferramentas de análise de código fonte, como Klocwork [5] Coverity [6], , Parasoft [7], Rational Software Analyzer [8], Fortify SCA [9], Checkmarx Suite [10], Code Secure [11], entre outras [42]. No entanto, estudos recentes têm introduzido ferramentas com inteligência de aprendizado de máquina. Contudo, essas ferramentas têm se concentrado principalmente na detecção de vulnerabilidades em códigos fonte, muitas vezes abordando apenas um tipo específico de vulnerabilidade e necessitando de especialistas[44],[47].

Uma outra ferramenta muito aceita e utilizado no mercado corporativo [48] é a ferramenta SonarQube. O SonarQube é uma ferramenta de análise estática de código que identifica *bugs*, vulnerabilidades e más práticas de programação [49]. Durante o desenvolvimento, os desenvolvedores escrevem e combinam o código, e o SonarQube, integrado ao processo de prática do desenvolvimento do software, avalia o código. Seu componente via Scanner faz uma leitura do código e envia os resultados para um servidor. A arquitetura do SonarQube é composta por três elementos principais:

[5] `https://www.klocwork.com/`
[6] `https://www.synopsys.com/software-integrity.html`
[7] `https://www.parasoft.com/`
[8] `https://www.ibm.com/products/rational-software-analyzer`
[9] `https://www.microfocus.com/en-us/fortify-integrations/visual-studio`
[10] `https://www.checkmarx.com/`
[11] `https://www.veracode.com/`

- O servidor web para configuração.
- O servidor de pesquisa baseado em ElasticSearch para buscas.
- O mecanismo de computação para armazenar relatórios e métricas.

O SonarQube utiliza *plug-ins* oficiais para avaliar e categorizar o código com métricas tradicionais. Essas métricas abrangem várias áreas, realizando medição da complexidade, identifica blocos de arquivos e linhas duplicadas, avalia os problemas de acordo com a gravidade, detecta cheiros de código e taxa de *Technical Debt* com base em avisos e alertas, contabilizando *bugs* e esforços de correção, detecta vulnerabilidades analisando métricas gerais e avalia cobertura e métricas de testes de unidade. As métricas são armazenadas em um banco de dados para rastrear mudanças ao longo do tempo. O SonarQube introduz a estratégia de Limpar o código por meio dos portões de qualidade . Isso direciona os usuários a usarem o novo código, assegurando sua qualidade e segurança. Os portões de qualidade são ajustados aos padrões do usuário, e um status verde indica aprovação. Ao analisar solicitações de mesclagem, os portões de qualidade garantem que o código atenda aos padrões antes da fusão. Os portões de qualidade também são visíveis na interface do SonarQube, permitindo a adição de metadados a solicitações de mesclagem com problemas do SonarQube, na figura 2.3, é possível observar com mais detalhes o funcionamento do sistema. Para uma explicação mais detalhada, recomendo também consultar o site do fornecedor SonarQube [12].

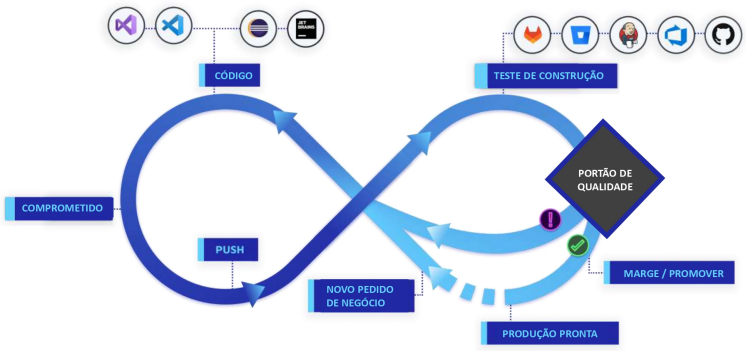


Figura 2.3: Visão Geral do SonarQUBE.
Fonte: Adaptado de SonarQUBE [50].

Considerando a ampla adoção do SonarQUBE no mercado e sua capacidade de abordar três dos cinco critérios de risco identificados, além do fato de que a ferramenta já foi previamente homologada e empregada pela empresa, requerendo apenas uma atualização

[12] `https://docs.sonarsource.com/sonarqube/latest/`

e verificação, essa ferramenta será utilizada neste estudo para validar os critérios de risco relacionados a Vulnerabilidades, *Bugs* e *Code Smells*.

Para identificar e validar o critério de risco relacionado ao uso inadequado de licenças, diversas ferramentas estão disponíveis. Entre as comercializadas com custo, destacam-se o FlexNet Code Insight [13], Black Duck [14], Fossa [15] e WhiteSource [16]. Algumas dessas ferramentas permitem instalação local, enquanto outras requerem o envio do código fonte para a nuvem, o que pode ser inviável. Neste estudo, para considerar o esforço, praticidade e precisão dos resultados, optou-se por utilizar a ferramenta FoSSology [17] para análise de licenças de código aberto. O FoSSology emprega um algoritmo de correspondência de padrões chamado Binary Symbolic Alignment Matrix (BSAM), inspirado em técnicas biomédicas.[51] Ele identifica 78 variações de licenças distintas [40], organizadas em uma hierarquia, como ilustrado na Figura 2.4.

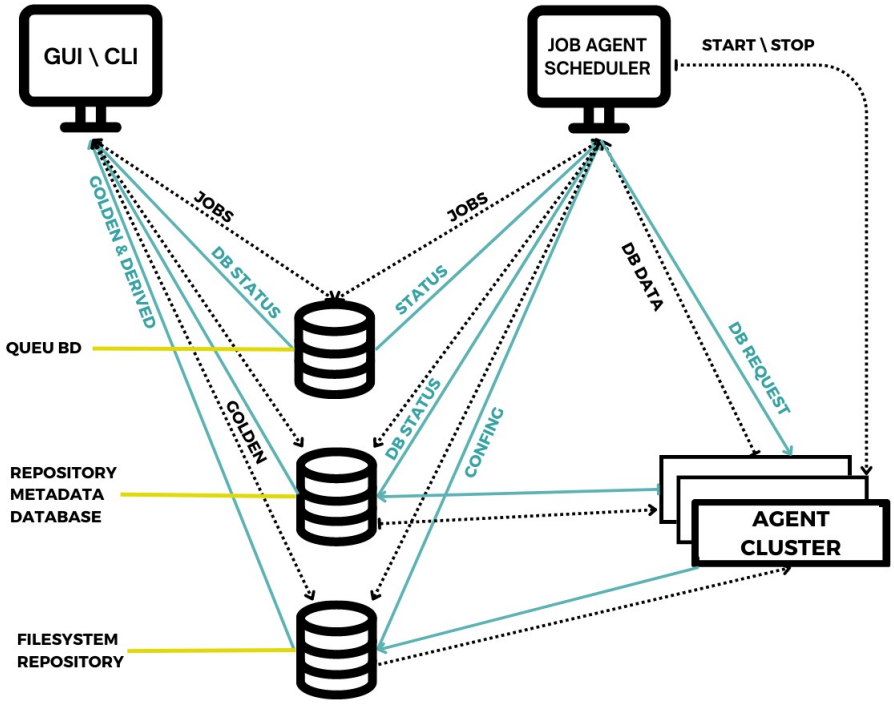


Figura 2.4: Visão Geral da FOSSology.
Fonte: Adaptado de Robert Gobeille [51].

O FOSSology possui um valor significativo como serviço de computação e no compartilhamento de resultados. Embora o próprio software não seja recente, é composto por um

[13] https://www.revenera.com/software-composition-analysis
[14] https://osbsoftware.com.br/produto/black-duck-softwarenone
[15] https://www.thoughtworks.com/pt-br/radar/tools/fossa
[16] https://www.mend.io/open-source-license-compliance/
[17] https://www.fossology.org

repositório, banco de dados e um conjunto de ferramentas. O processo envolve o carregamento de software no repositório via script ou interface web. O agendador FOSSology executa agentes que extraem arquivos de componentes de contêineres e realizam análises específicas, armazenando os resultados. Esses resultados são acessíveis para geração de relatórios ou uso por outros agentes de análise.

Para identificar e analisar o critério de risco de plágio ou similaridade no software será considerado a ferramenta CopySpider pela sua flexibilidade de instalação e por sua boa acurácia em comparar códigos semelhantes na internet. A ferramenta teve sua origem como parte da pesquisa do trabalho de iniciação científica realizado por Clever Marcos Teixeira, orientado pelo Professor Marcelo Augusto Cicogna, nos anos de 2011 e 2012. [52]. Em 2013, Clever e Marcelo, os mencionados autores, tornaram-se sócios e colaboraram no aprimoramento da ferramenta. O CopySpider [18] oferece uma licença de apoiador, proporcionando recursos específicos para aumentar a produtividade na avaliação de grandes volumes de arquivos em lotes. O processo é simples. Você envia seus arquivos em lote para o CopySpider, que os compara com outros arquivos na internet, fornecendo um resultado detalhado da pesquisa. A imagem 2.5 ilustra o funcionamento desse processo. Além disso, o CopySpider é reconhecido por sua eficácia na identificação de similaridades e plágios, sendo uma ferramenta boa para garantir a integridade e autenticidade dos documentos.

[18] `https://copyspider.com.br/main/pt-br/history.`

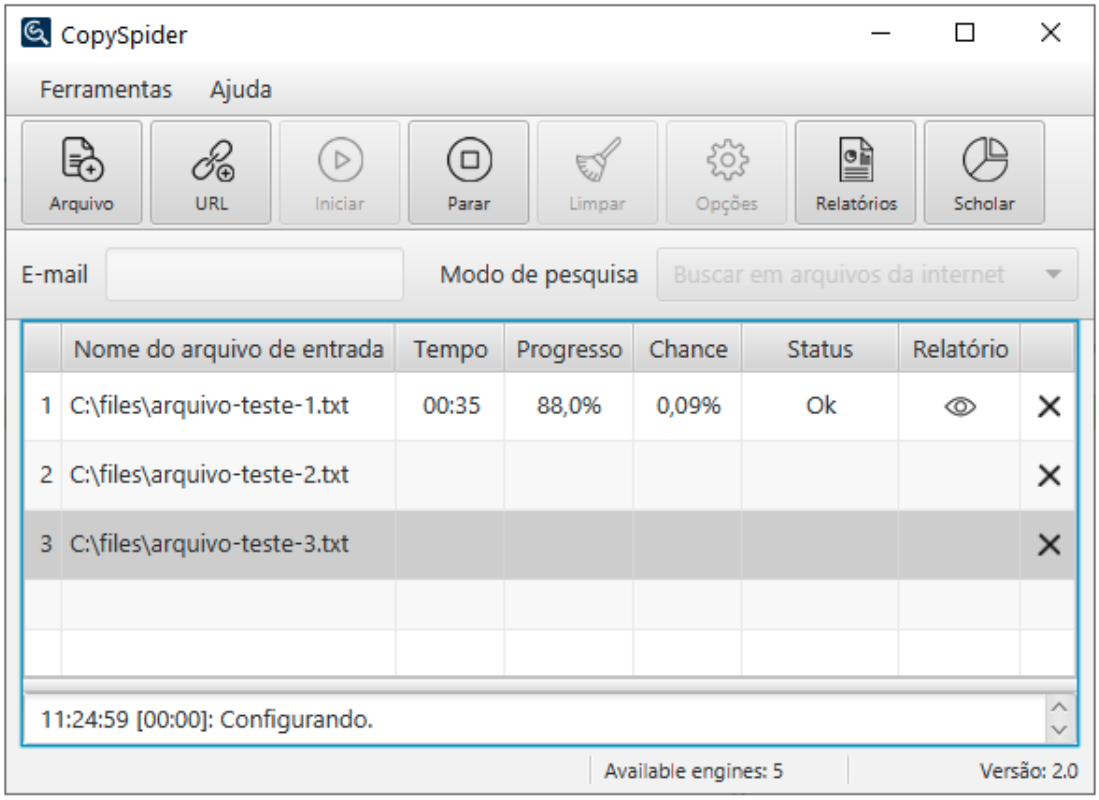


Figura 2.5: Visão Geral CopySpider.
Fonte: Site Oficial CopySpider[53].

2.3 Critério de Risco

Conforme preconizado pela ABNT ISO GUIA 73:2009 [54], o Critério de Risco (CR) desempenha o papel de uma referência, representando um parâmetro estabelecido ou uma expectativa que orienta a avaliação ou a tomada de decisão relativa à magnitude de um risco. Estes critérios podem incorporar variáveis como custos, vantagens, requisitos legais, preferências ou prioridades das partes envolvidas.

2.4 Modelo de Processo de Apoio à Decisão

O Modelo de Processo de Apoio à Decisão (MCDA), também conhecido como Múltiplos Critérios de Tomada de Decisão (MCDM), é uma abordagem que explora explicitamente múltiplos critérios para embasar decisões, com suas raízes na pesquisa operacional e aplicação em diversos setores, incluindo agricultura, gestão de recursos e cuidados de saúde [55]. O Processo de Apoio à Decisão envolve a criação de um sistema de preferências, onde as alternativas são avaliadas em relação aos critérios escolhidos para o modelo em

desenvolvimento. Isso visa a recomendação e validação das soluções possíveis, preparando para a próxima etapa, conforme ilustrado na Figura 2.6.

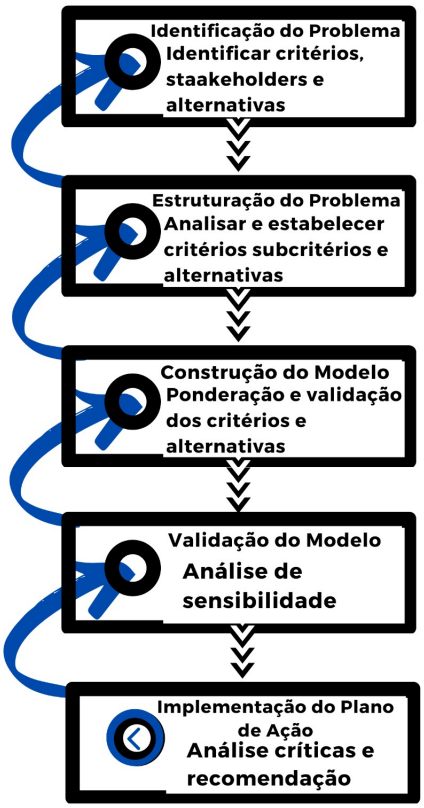


Figura 2.6: Etapas do Processo de Apoio à Decisão.
Fonte: Adaptada de Belton e Stewart [56].

Existem três categorias principais de métodos de tomada de decisão multicritério [57]:

- Modelos de medição de valor: Exemplos incluem (AHP) e teoria da utilidade de múltiplos atributos (MAUT).
- Modelos de Meta, Aspiração e Referência: Englobam métodos como a programação por meta e (TOPSIS).
- Métodos de superação: Incluem famílias como (ELECTRE) e (PROMETHEE).

O Processo de Análise Hierárquica (em inglês, Analytic Hierarchy Process - AHP), desenvolvido inicialmente por Saaty [58], é uma abordagem que integra as opiniões de especialistas e as pontuações de avaliação em um sistema hierárquico simples, decompondo problemas complexos em superiores para inferiores. O (AHP) facilita a tomada de decisão ao analisar a distribuição e os efeitos das multivariáveis em um contexto específico, organizando os critérios em hierarquia para alinhá-los aos objetivos e resolver problemas

complexos [59]. O Processo de Rede Analítica (em inglês, Analytic Network Process - (ANP), mais abrangente que o (AHP),é especialmente indicado para decisões de longo prazo, uma vez que permite a incorporação de loops de realimentação [60].

Em casos de muitas comparações de pares, o (TOPSIS) é uma ferramenta de decisão multi-atributo reconhecida por sua eficácia [61]. Ele busca alternativas que não apenas estejam próximas da solução ideal positiva, mas também distantes da solução ideal negativa [57].

O método (ELECTRE), introduzido por Benayoun et al.[62], incorpora conceitos de concordância, discordância e classificação originados de cenários reais, sendo amplamente aplicado em áreas como energia e finanças [63], [64].

O (PROMETHEE), desenvolvido por Brans [65] e ampliado por Vincke e Brans [66], é um método (MCDA) mais recente. Ele aborda a classificação de alternativas baseada em critérios, incluindo os conflitantes, e se destaca por sua simplicidade tanto em design quanto aplicação [67].

Com o objetivo de simplificar a complexidade de problemas hierárquicos e adotar uma abordagem de curto prazo, este estudo opta pelo método (AHP). No entanto, reconhecendo a necessidade de refletir as opiniões dos especialistas de maneira mais abrangente e considerando a natureza complexa do problema, o método Fuzzy será incorporado ao modelo. Isso permitirá uma análise mais completa e precisa das avaliações dos especialistas. O método Fuzzy, desenvolvido por Zadeh [68], permite lidar com as impressões subjetivas dos julgamentos humanos por meio da teoria dos conjuntos fuzzy. Com a combinação do (AHP) e do Fuzzy [69],[70], [71], será possível resolver sistematicamente o problema de seleção, utilizando os conceitos da teoria dos conjuntos fuzzy e análise hierárquica. O método (FAHP), essencialmente, representa uma abordagem (AHP) tradicional no domínio fuzzy, empregando números fuzzy para processamento em vez de números reais [72]. As primeiras aplicações do (FAHP) incluem Van Laarhoven e Pedrycz [73], que usaram funções de pertinência triangulares para comparações de pares. Buckley [74] avançou o campo ao determinar prioridades nebulosas usando funções de pertinência triangulares. Chang [75] também introduziu um método com números triangulares em comparações pareadas. Neste estudo, são implementados os métodos de Buckley [74] para determinar os pesos de importância relativa de critérios e alternativas. No (FAHP), as comparações são realizadas em pares, tanto dos critérios quanto das alternativas, são realizadas utilizando variáveis linguísticas representadas por números triangulares [76], conforme ilustrado na tabela 2.1.

Tabela 2.1: Termos Linguísticos e os Números Fuzzy Triangulares correspondentes.

Escala Saaty	**Termo Linguístico**	**Escala Triangular Fuzzy**
1	Igualmente importante	(1,1,1)
3	Pouco importante	(2,3,4)
5	Bastante importante	(4,5,6)
7	Fortemente importante	(6,7,8)
9	Absolutamente importante	(9,9,9)
2,4,6,8	Valores intermediários	(1,2,3),(3,4,5),(5,6,7),(7,8,9)

Com base nos números *fuzzy* triangulares correspondem a esses termos linguísticos, se o tomador de decisão declara que, Critério 1 (C1) é pouco importante em relação ao Critério 2 (C2), então a escala triangular fuzzy seria representada como (2,3,4). Em contraste, na matriz de contribuição par a par dos critérios, a comparação de C2 com C1 assumirá a escala triangular fuzzy como (1/4,1/3,1/2). Os sete passos para a aplicação do FAHP são detalhados a seguir:

Primeiro Passo: É apresentado a matriz de contribuição entre pares do decisor, que é representada pela Equação 1. Nesta equação, $\widetilde{d}_{ij}^{k}$ significa a inclinação do tomador de decisão k^{th} para o critério I^{th} sobre o critério j^{th}, usando números triangulares difusos. O símbolo (~)representa a demonstração de um número triangular. Por exemplo, no exemplo dado, $\widetilde{d}_{12}^{1}$ representa a preferência do primeiro tomador de decisão do primeiro critério sobre o segundo critério, e é igual a $\widetilde{d}_{12}^{1} = (2, 3, 4)$.

Equação 1:

$$\widetilde{A}^{K} = \begin{bmatrix} \widetilde{d}_{11}^{k} & \widetilde{d}_{12}^{k} & \cdots & \widetilde{d}_{1n}^{k} \\ \widetilde{d}_{21}^{k} & \widetilde{d}_{22}^{k} & \cdots & \widetilde{d}_{2n}^{k} \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{d}_{n1}^{k} & \widetilde{d}_{n2}^{k} & \cdots & \widetilde{d}_{nn}^{k} \end{bmatrix}$$

Segundo passo: Quando vários tomadores de decisão estão envolvidos, suas preferências individuais, conhecidas como $\widetilde{d}_{ij}^{k}$, são combinadas e usadas para calcular $\widetilde{d}_{ij}$ usando a Equação 2.

Equação 2:

$$\widetilde{d}_{ij} = \frac{\sum_{k=1}^{k} \widetilde{d}_{ij}^{k}}{k}$$

Terceiro passo: Com base nas preferências médias, a matriz de contribuição aos pares é modificada conforme representado na Equação 3.

Equação 3:

$$\widetilde{A} = \begin{bmatrix} \widetilde{d}_{11} & \cdots & \widetilde{d}_{1n} \\ \vdots & \ddots & \vdots \\ \widetilde{d}_{n1} & \cdots & \widetilde{d}_{nn} \end{bmatrix}$$

Quarto passo: Buckley [74] propõe calcular a média geométrica dos valores de comparação difusa para cada critério usando a Equação 4. Deve-se notar que $\widetilde{r}_i$ continua a denotar valores triangulares.

Equação 4:

$$\widetilde{r}_i = \left(\prod_{j=1}^{n} \widetilde{d}_{ij} \right)^{\frac{1}{n}}, i - 1, 2, ..., n$$

Quinto passo: Os pesos para cada critério podem ser determinados utilizando a Equação 5, que envolve os seguintes passos subsequentes:

Calcule o vetor de soma para cada $\widetilde{r}_i$. Calcule o inverso (elevado à potência de -1) do vetor de soma. Ajuste o número triangular difuso para organizá-lo em ordem crescente. Para encontrar o peso difuso do critério i, multiplique cada $\widetilde{r}_i$ por este vetor invertido, denotado como $\widetilde{\omega}_i$.

Equação 5:

$$\widetilde{\omega}_i = \widetilde{r}_i \left(\widetilde{r}_1 + \widetilde{r}_2 + ... + \widetilde{r}_n \right)^{-1} = (l\omega_i, m\omega_i, u\omega_i)$$

Sexto passo: Como $\widetilde{\omega}_i$ permanece na forma de números triangulares difusos, eles devem passar por um processo reverso do processo *fuzzy* usando o método do Centro de Área, conforme sugerido por Chou e Chang [77]. Isso envolve a aplicação da

Equação 6:

$$M_i = \frac{(l\omega_i + m\omega_i + u\omega_i)}{3}$$

Sétimo passo: M_i é um valor numérico não difuso, mas requer normalização aplicando a Equação 7.

Equação 7:

$$N_i = \frac{M_i}{\sum_{i=1}^{n} \mathrm{m}_i}$$

Esses sete passos são executadas para determinar os pesos normalizados tanto dos critérios quanto das alternativas. Em seguida, ao multiplicar os pesos atribuídos a cada alternativa pelos respectivos critérios, as pontuações para cada alternativa são calculadas. Com base nesses resultados, a alternativa com a pontuação mais alta é recomendada ao tomador de decisão.

2.5 Conclusão Parcial

Neste capítulo, o objetivo de responder às seguintes perguntas foram alcançadas:

- (1) O que é auditoria de código fonte? Resposta: foi possível explorar que é uma análise detalhada do código de um software para garantir qualidade, segurança e conformidade.
- (2) Quais são os critérios de risco envolvidos? Resposta: foram identificados vulnerabilidades, *Bugs*, *Code Smell*, Plágio ou similaridade de código e uso indevido de licença.
- (3) Quais são os métodos e ferramentas usados na auditoria de código fonte? Resposta: Foram explorados métodos estáticos, dinâmicos e algumas ferramentas que possibilitou selecionar as ferramentas SonarQUBE, Fossology e CopySpider para validar e tratar os critérios de riscos.
- (4) Há métodos de apoio à decisão? Resposta: Sim, foram exploradas algumas técnicas e métodos de (MCDA), que resultou na escolha do método FAHP para priorização de riscos.

  No próximo capítulo, serão apresentados os trabalhos relacionados para aprofundar a compreensão e embasar as escolhas dos métodos, técnicas e ferramentas selecionados.

Capítulo 3

Trabalhos Relacionados

Com o intuito de realizar uma revisão bibliográfica, a presente pesquisa adotou o protocolo (PRISMA), conforme mencionado em [78]. Essa metodologia foi empregada para identificar as tendências e direcionamentos na área de interesse, contribuindo para a abordagem criteriosa e sistemática da literatura pertinente.

A coleta de informações foi primordialmente concentrada nas plataformas acadêmicas *Web of Science* e *Scopus*, com o propósito de apresentar uma revisão sistemática das metodologias e aplicações relacionadas a técnicas, métodos, ferramentas, risco e modelos de auditoria de código fonte. A busca na literatura foi conduzida por meio da utilização de diversas palavras chave, tais como: (Source code), (Source code vulnerability), (Source code bug), (Code smell), (Source code plagiarism), (Source code similarity), (Source code ownership), (Open license), (Open source license), (Source code audit software), (Source code audit), (Source code review), (Multicriteria analysis), (Multicriteria decision making), (Analytic hierarchy process), (AHP), (FAHP), (AHP-Fuzzy), (SonarQUBE), (FoSSology), (CopySpider), (Risk), (Risk management) e (Risk criterion), conforme ilustrado nas *Querys*:

- *Web of Science* - (TS=("source code"OR "source code vulnerability"OR "source code bug"OR "code smells"OR "source code plagiarism"OR "source code similarity"OR "code ownership"OR "Open license"OR "code license"OR "open source license"OR "source code audit"OR "code review") AND (TS=("multicriteria analysis"OR "multicriteria decision making"OR "analytic hierarchic process"OR "AHP"OR "FAHP"OR "AHP-Fuzzy"OR "SonarQUBE"OR "FoSSology"OR "copyspider"OR "risk criterion" OR "risk management"OR "risk"))).
- *Scopus* - TITLE-ABS-KEY(("source code"OR "source code vulnerability"OR "source code bug"OR "code smells"OR "source code plagiarism"OR "source code similarity"OR "code ownership"OR "open license"OR "code license"OR"open source li-

cense"OR "source code audit"OR "code review") AND ("multicriteria analysis"OR "multicriteria decision making"OR "analytic hierarchy process"OR "AHP"OR "FAHP" OR "ahp-fuzzy"OR "sonarqube"OR "fossology"OR "FoSSology"OR "copyspider"OR "risk criterion"OR "risk management"OR"risk")).

A adoção desse enfoque possibilitou a obtenção dos resultados, permitindo a aplicação da metodologia (PRISMA) para refinar de maneira mais precisa os artigos relacionados, conforme visualizado na figura 3.1. Esse processo estruturado permitiu a identificação e análise crítica dos estudos relevantes na área de auditoria de código fonte.

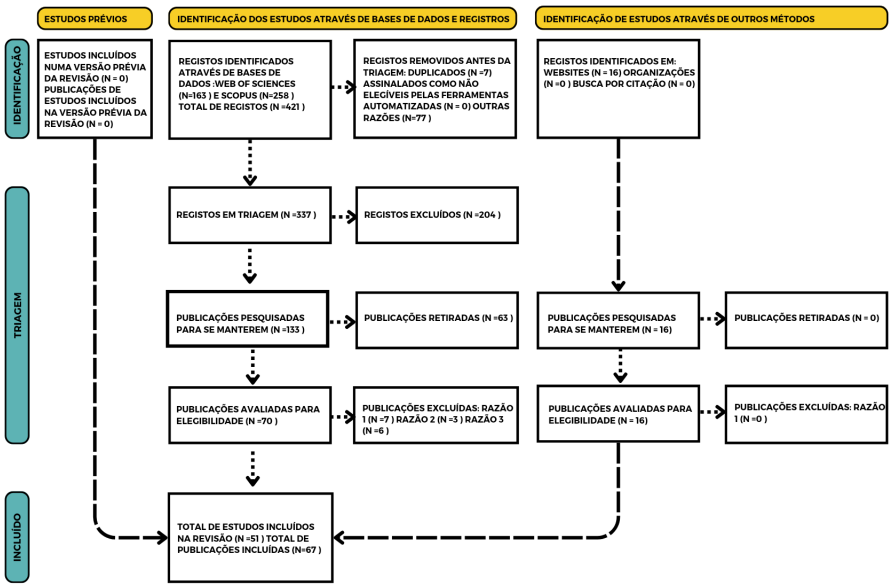


Figura 3.1: PRISMA 2020 Fluxograma Revisão Sistemática da Literatura.
Fonte: Adaptado de Page(2021) [78]

.

O número total de artigos identificados nas bases de dados selecionadas atingiu o total de 421 registros. Este estudo realizou uma seleção direcionada para incluir artigos de revistas acadêmicas e conferências, enquanto excluiu outros tipos de documentos, como livros e revistas. Além disso, foi aplicado um critério de idioma, restringindo a seleção apenas para artigos redigidos em língua inglesa. Outro critério relevante adotado foi a delimitação temporal, abrangendo o período de 2013 a 08/2023. Após remover artigos duplicados e organizar com ajuda da ferramenta Rayyan [1], o total de artigos foi reduzido para 337.

[1] `https://www.rayyan.ai/`

Na etapa inicial de triagem, foi realizado uma análise dos títulos e resumos dos artigos. Isso resultou na exclusão de 204 artigos que não estavam alinhados com os objetivos deste estudo, pois abordavam questões como vulnerabilidade de redes, riscos associados à infraestrutura e outros ativos de TI (Tecnologia da Informação). Culminando em um conjunto de 133 artigos restantes.

Posteriormente, os 133 artigos selecionados foram submetidos a uma leitura completa e detalhada. Desse grupo, 63 artigos foram descartados por não contribuírem de maneira significativa para o escopo deste estudo. Alguns desses artigos tratavam de modelos de apoio à decisão que não estavam relacionados a códigos fonte, enquanto outros abordavam temas como prevenção de riscos e plágio ou similaridade de textos, e não de códigos fonte. Isso resultou em um conjunto final de 70 artigos que atenderam aos critérios de relevância e pertinência estabelecidos.

Uma segunda avaliação foi realizada sobre os 70 artigos restantes, identificando que 19 deles estavam relacionados à auditoria de sistemas mobile e Android, que estão além do escopo deste estudo definido como Razão1. Além disso, o conjunto denominado Razão2 tratava de estratégias para prevenir erros e vulnerabilidades, enquanto o Razão3 abordava técnicas para melhorar os testes de software com o intuito de prevenir problemas. Ambos os conjuntos não estavam em consonância com o foco desta pesquisa, resultando na exclusão desses artigos. Como resultado, um total de 51 artigos permaneceram após essa etapa de seleção.

Adicionalmente, este estudo também considerou a inclusão de dados estatísticos de mercado, informações sobre aumento de problemas de qualidade e segurança, assim como abordagens, ferramentas, técnicas e métodos para aprimorar a identificação de problemas de segurança e qualidade no software. Esta inclusão resultou em um acréscimo de 16 recursos de Websites, culminando, portanto, em um conjunto final de 67 artigos relacionados ao tema central do estudo, conforme ilustrado na tabela 3.1

Tabela 3.1: Tabela de Artigos Relacionados.

Referencial Teórico	**Categoria**
Wang (2021) [79], Zhang (2022) [80] Abuhassan (2022) [81], Schreiber (2021) [82] Filus (2021) [83], d'Aragona (2023) [84] Do Xuan (2023) [85], Singh (2021) [86], Pang (2023) [87] Woo e Park (2021) [88],Liu (2019) [89] Wang (2022) [90], Saborido (2022) [91] HegedHus (2022) [92], Liu (2022) [93] Xu, Sihan, e Gao (2023) [94],Moraes (2021) [95] Higashi (2016) [96], Novak (2021) [97], Hirsh (2021) [98] karnalim (2022) [99], akremi (2021) [100] Kapitsaki (2016)[101], Kapitsaki (2019) [102] Gamalielsson (2017) [103], Vendome (2015)[104] Vendome (2017)[105], Caneill (2017) [106] Serafini (2022) [107], Qiu, Shi e German (2021) [108] Karnalim (2022) [109],Karnalim (2021) [110] Hrkut (2023) [111], Anjali (2015) [112], Nazim (2022) [113] Qui (2020) [114], Jin (2022) [115], Jain (2021) [116] Algarni (2022) [117], Papoutsoglou (2022) [118]	Vulnerabilidades, *Code Smells*, *Bugs* Uso Indevido de Licença, Plagio ou Similaridade, Métodos, Técnicas e Ferramentas
Smith (2016) [119],Yang (2015) [120]	AHP
Yang (2015) [120], Cui (2020) [121]	BPM
Anjum (2020) [122], Song (2016) [123]	TOPSIS
Abushark (2022) [124], Agrawal (2020) [125]	FUZZY
Seh (2022) [126], Bhatt (2022) [127]	MCDA
Goyal (2017) [128]	MCDM
MarketReport [1], GrandViewResearch [2] ABES [3], CSO [4], Citibank [13], DAO [14] Ransomware WannaCry [15], IBM [16] OWASP [28], LGPD [129], Fossology [130], Rayyan [131] CopySpider [53], SonarQUBE [50], Valdecy [132] Positive Technologies [27]	Website

No contexto da abordagem a vulnerabilidades, *code smells* e *bugs* em software, a literatura oferece uma coleção de estudos que exploram a complexidade dessas questões. Por exemplo, Wang (2021) [79], Zhang (2022) [80] e Abuhassan (2022) [81], Algarni (2022) [117], Hirsh (2021) [98], ressaltam a crescente urgência em lidar com a expansão acelerada

de vulnerabilidades, enfatizando a necessidade de abordagens eficazes respaldadas por métodos, técnicas e ferramentas adequadas. Esses estudos destacam, ademais, que a combinação nascente entre aprendizado de máquina e análise estática poderia ser uma rota promissora, ainda que careça de aperfeiçoamento.

No mesmo âmbito, Schreiber (2021) [82], Do Xuan (2023) [85], akremi [100], Jin (2022) [115], destaca a relevância de ferramentas intuitivas para permitir que os desenvolvedores identifiquem e tratem problemas de forma mais eficiente. Filus (2021) [83], Nazim (2022) [113], ressalta que vulnerabilidades, *bugs* e *code smells* podem acarretar consequências graves, desde violações de segurança até prejuízos financeiros e deterioração de reputação, e que abordar esses desafios exige esforços de pesquisa e uma melhora na cultura de segurança. O estudo ainda aponta indicadores, como aceleração no ciclo de desenvolvimento de software, recursos de teste limitados e falta de experiência em segurança entre programadores, que podem desencadear tais problemas.

A detecção dessas problemáticas é enfatizada em diversos estudos, sendo respaldada pelo emprego de ferramentas e técnicas especializadas. Por exemplo, Liu (2019) [89] conduziu uma pesquisa abordando essa temática, e em um subsequente trabalho de Liu (2022) [93], juntamente com d'Aragona (2023) [84], a aplicação de ferramentas como o SonarQube, aliada a outras técnicas, permitiu a identificação de questões relacionadas a vulnerabilidades, bugs e code smells. Wang (2022) [90], por sua vez, enfatiza a relevância de ferramentas estáticas como o SonarQube e o Findbugs para a identificação desses problemas.

É evidente que, no contexto atual, onde o software permeia praticamente todos os aspectos de nossa vida, desde navegar na internet até acessar serviços críticos como saúde e bancos, os desenvolvedores enfrentam um cenário desafiador. Saborido (2022) [91] e HegedHus (2022) [92] destacam que a demanda para resolver questões de segurança e problemas em software frequentemente supera o tempo disponível para a evolução do software. Eles sublinham a ampla adoção do SonarQube na indústria como uma ferramenta que pode fornecer resultados significativos quando configurada corretamente. A literatura, assim, consolida a importância das abordagens para aprimorar a qualidade e a segurança do software em face do ambiente tecnológico atual.

Uma outra preocupação de risco investigada é a complexidade associada ao uso inadequado de licenças de software e à má utilização de bibliotecas de terceiros em contextos de desenvolvimento de projetos. Por exemplo, em suas respectivas pesquisas, Xu, Sihan, e Gao (2023) [94], Higashi, Yunosuke, e Manabe (2016) [96], Kapitsaki (2016)[101], (2019) [102], e Gamalielsson (2017) [103], Qui (2020) [114], Papoutsoglou (2022) [118], exploraram métodos, técnicas e ferramentas automatizadas para abordar a categorização e análise das diferentes licenças de software, enfatizando a necessidade de uma abordagem

precisa e escalável nesse processo.

Além disso, o trabalho conduzido por Vendome (2015)[104] e (2017)[105] adentrou nos meandros da tomada de decisão dos desenvolvedores em relação às escolhas e eventuais alterações de licenciamento do software. Este estudo revelou que muitas vezes a consideração sobre licenciamento tende a se perder no decorrer do projeto, apesar de sua relevância na garantia da conformidade legal e da integridade do código fonte. Complementando essa abordagem, a pesquisa de Caneill (2017) [106] explorou as tendências emergentes na adoção de código aberto e software livre, contribuindo para a compreensão das dinâmicas envolvidas nesse cenário.

Adicionalmente, o estudo conduzido por Woo e Park (2021) [88] aprofundou-se na identificação precisa e escalável da reutilização de licenças, inclusive modificações, que poderiam acarretar em vulnerabilidades de segurança e violações de licença. A investigação de Serafini (2022) e Qiu, Shi e German (2021) [107, 108] propôs abordagens metodológicas para a determinação precisa dos tipos de licenças, reforçando a importância crítica de examinar atentamente as licenças antes de incorporá-las em um software.

Por fim, o estudo realizado por Moraes (2021) [95] lançou luz sobre a problemática da quantidade média de licenças embarcadas em projetos de software, revelando que muitos projetos apresentam problemas decorrentes do uso inadequado de licenças, ao mesmo tempo em que os profissionais envolvidos demonstram falta de conscientização sobre as implicações da utilização indevida de licenças. Essas investigações fornecem uma visão abrangente sobre as questões associadas ao licenciamento de software e à utilização de bibliotecas de terceiros, contribuindo significativamente para a compreensão das melhores práticas e riscos nesse contexto.

Um outro desafio é o problema de plágio ou similaridade de código fonte, cujas implicações podem ser amplas e severas. Tal prática, que alguns desenvolvedores vêm adotando, pode resultar em sérios contratempos para as organizações, incluindo, mas não se limitando a, violações de direitos autorais e prejuízos financeiros. Tais códigos comprometem não somente a integridade do software, mas também a segurança do mesmo, engendrando vulnerabilidades, *bugs* e *code smells* que deterioram tanto a qualidade quanto a segurança do software.

Exemplificativamente, estudos como o de Karnalim (2022) [109], Karnalim (2021) [110], Hrkut (2023) [111] e Anjali (2015) [112] , Singh, Malya and Gupta, Vishal (2021) [86], Pang (2023)[87], Novak (2021) [97], karnalim (2022)[99], Jain (2021) [116] elucidam a complexidade e importância da identificação de fontes de código similares ou plagiados. Nesse sentido, percebe-se a necessidade imperativa de recorrer a métodos e ferramentas especializadas para apoiar o processo de identificação e resolução desse problema intrincado. A literatura realça a importância de lidar ativamente com essa ameaça, a fim de

salvaguardar tanto a integridade quanto a reputação das organizações e seus softwares.

A literatura destaca a complexidade inerente à gestão de riscos em projetos de softwares, ressaltando que identificar os riscos é apenas uma etapa preliminar, sendo igualmente crucial estabelecer um modelo eficaz para priorizar ações de tratamento desses riscos. Em cenários de elevada complexidade, onde a tomada de decisão humana pode ser insuficiente, abordagens como a análise multicritério têm emergido como alternativas fundamentadas. Estudos como o de Smith (2016) [119] exemplificam a utilização do método de Análise Hierárquica (AHP) para comparar projetos sob diversos critérios de qualidade, revelando prioridades claras como manutenibilidade, usabilidade, reusabilidade e desempenho.

Em pesquisa de Yang (2015) [120] e Cui (2020) [121], o método (AHP) foi empregado para criar um modelo capaz de classificar perdas econômicas de empresas resultantes da exploração de vulnerabilidades, apresentando resultados promissores. Em outro contexto, Anjum (2020) [122] comparou (AHP) com o Método de Ponderação Bayesiana (BPM) para priorizar riscos de vulnerabilidade, enquanto Song (2016) [123] combinou (AHP) e a técnica Fuzzy para avaliar a gravidade de vulnerabilidades.

Abushark (2022) [124], Agrawal (2020) [125] e Seh (2022) [126] investigaram a priorização e seleção de atributos em sistemas de detecção de intrusão, combinando AHP, a criação hierárquica da estrutura TOPSIS para ordenação de preferências e conjuntos Fuzzy, considerando a incerteza das decisões especializadas. Em tratativas de vulnerabilidades e bugs, Bhatt (2022) [127] empregou abordagens fuzzy em combinação com o método TOPSIS para classificar scanners de vulnerabilidades, obtendo resultados positivos. O Goyal (2017) [128] utilizou o (AHP) para priorizar a triagem de bugs. Esses estudos demonstram a crescente adoção de métodos multicritério na priorização de riscos e tratamento em projetos de desenvolvimento de software, auxiliando na tomada de decisão.

3.1 Conclusão Parcial

Neste capítulo, foi atingido o objetivo de examinar os estudos relacionados, contou também com ajuda do protocolo (PRISMA) para realizar uma triagem e selecionar os artigos para a composição deste trabalho. Durante essa análise, foi perceptível que existem diversas técnicas, métodos e ferramentas disponíveis para identificar riscos como vulnerabilidades, *bugs*, *code smell*, plágio ou similaridade de código fonte, bem como o uso indevido de licenças. No entanto, ficou evidente que esses estudos tratam dos critérios de risco de forma isolada, o que pode resultar em uma abordagem fragmentada para empresas que desejam abordar holisticamente todos os fatores de risco em seus softwares.

Além disso, identificou-se a existência de modelos de tomada de decisão que auxiliam na classificação das abordagens de mitigação de riscos. No entanto, esses modelos também tendem a focar em um tipo específico de risco, o que pode não atender completamente às necessidades de uma empresa que busca uma abordagem abrangente.

Diante desse panorama, o próximo capítulo se propõe a aplicar técnicas e ferramentas que analisam os cinco critérios de risco em um software de mercado. Além disso, será empregado um modelo multicritério para avaliar e priorizar os riscos. O intuito é fornecer uma abordagem mais integrada e embasada para empresas que desejam identificar, classificar e tratar os diversos fatores de risco de forma coordenada, visando aprimorar a conformidade, qualidade e segurança de seus produtos de software.

Capítulo 4

Estudo de Caso

Com a adoção do método ágil no desenvolvimento do software, o número crescente de novas solicitações dos clientes e os prazos cada vez mais curtos têm comprometido o processo atual de revisão de código fonte. Diante desse cenário, este estudo de caso tem como objetivo aplicar métodos e ferramentas automatizadas no software para validar a existência dos critérios de riscos que foram identificados na Revisão Sistemática da Literatura.(RSL) Além disso, busca-se aplicar o método (FAHP) para nortear em qual módulo do software possui o maior risco, apresentando um modelo sequencial de priorização. O modelo será validado com a participação de especialistas, fornecendo um panorama atual que subsidiará o plano de ação para resolver os problemas complexos identificados e auxiliar no desenvolvimento de estratégias.

4.1 Aplicação do Método e Ferramenta de Auditoria de Código Fonte

4.1.1 Cenário da Empresa e Software ATIVUControl

Com mais de três décadas de atuação, a empresa Ativu Tecnologia se destaca no mercado de tecnologia, especializada em fornecer soluções que otimizam a governança e gestão dos gastos relacionados a Telecom, Tecnologia da Informação e Facilites para seus clientes. Seu software, originalmente conhecido como TEMControle e agora denominado ATIVUControl, foi desenvolvido utilizando a linguagem de programação Java e adota uma arquitetura monolítica composta por nove módulos distintos. Após o processo de compilação, o sistema fica acessível através de navegadores web. Diante desse cenário, a empresa Ativu Tecnologia, com uma extensa base de dados de clientes, com mais de 326GB de informações, sendo 195.901 de dados sensíveis de usuários destes clientes, precisa estar atenta à qualidade e segurança do código fonte de seu software. Identificar

e tratar problemas em um software com muitos clientes é uma tarefa complexa e crucial para evitar danos à empresa. Essa estrutura é ilustrada em detalhes na figura 4.1, demonstrando a interconexão entre os módulos e suas dependências.

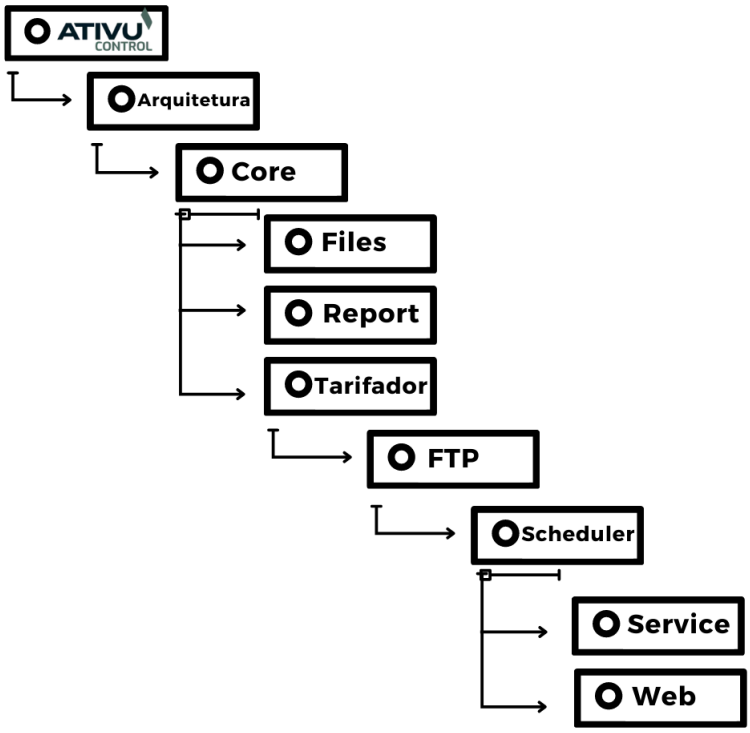


Figura 4.1: Software ATIVUControl - Arvore de Dependência.
Fonte: Elaboração própria.

A importância dos módulos e suas interdependências ressalta a necessidade de um funcionamento harmonioso para alcançar os resultados desejados no software. Essa complexidade de priorização em incidentes de risco enfatiza a exigência de um modelo com técnicas, métodos e ferramentas para auxiliar nesse processo. Na Tabela 4.1, é apresentado o propósito e as funcionalidades de cada módulo.

Tabela 4.1: Propósito e Funcionalidades dos Módulos no Software ATIVUControl.

Software ATIVUControl	**Propósito e Funcionalidades**
Arquitetura	Módulo que contem interfaces para a construção de funcionalidades.
Core	Módulo que contem regras de negócio relacionadas ao banco de dados.
Files	Módulo que contem funcionalidades para acesso e extração de conteúdos e criação de arquivos.
FTP	Módulo que contem funcionalidades para acesso a ftp, sftp, sistema de arquivos e coleta de arquivos.
Report	Módulo que contem regras de negócio para geração de relatórios.
Scheduler	Módulo que contem regras de negócio para agendamento de tarefas.
Service	Módulo que contem o serviço ativucontrol para processamento assíncrono.
Tarifador	Módulo que contem regras de negócio para processamento de dados de arquivos como bilhetes CDR, Impressoras e MDM.
Web	Módulo que contem as páginas de acesso via web para publicação na internet.

Atualmente, o software ATIVUControl desempenha um papel fundamental ao atender uma ampla gama de clientes e CNPJs ativos. Hospedado em seu datacenter, esse sistema armazena uma vasta quantidade de dados confidenciais e sensíveis pertencentes aos funcionários dos clientes. Esse ambiente é de vital importância, porém também é inerentemente suscetível a riscos. Na tabela 4.2 é possível visualizar o cenário atual da empresa e do software ATIVUControl.

Tabela 4.2: Cenário da Empresa e Software ATIVUControl.

Software ATIVUControl	**Quantidade/Tamanho**
Cliente/CNPJ	247
Estrutura módulo	9
Código fonte em disco	1GB
Arquivo.java	32.713
Pastas	2.696
Linhas de código	900.000
Bibliotecas Java	275
Banco de dados transacional	326GB
Banco de dados dimensional	613GB
Dados pessoais/sensíveis	195.901

4.1.2 Validação dos Critérios de Risco de Vulnerabilidade, *bugs* e *code Smell* a partir do software SonarQUBE

Para validar os critérios de riscos referentes a vulnerabilidades, *bugs* e problemas de *Code Smells*, foi empregado a ferramenta SonarQube. Esta está hospedada na mesma instância que abriga o repositório do código fonte do software AtivuControl no GitHub [1], desenvolvido usando o ambiente Eclipse [2]. Essa instância é equipada com um processador de núcleo único e quatro threads, operando a 3.2 GHz, 16 GB de RAM e 1 TB de armazenamento. Ao empregar práticas de Integração e Entrega Contínua com o Jenkins [3], foram criadas tarefas específicas para cada módulo do sistema, responsáveis por executar o *Scanner* correspondente e coletar as informações. Nesse contexto, a execução dos nove *Scanners* demanda cerca de treze minutos no total, gerando resultados em média de um minuto e meio por tarefa de *Scanner*. A estrutura do ambiente SonarQUBE na empresa é melhor ilustrada na Figura 4.2.

[1] `https://github.com/`
[2] `https://https://eclipseide.org/`
[3] `https://www.jenkins.io/doc/`

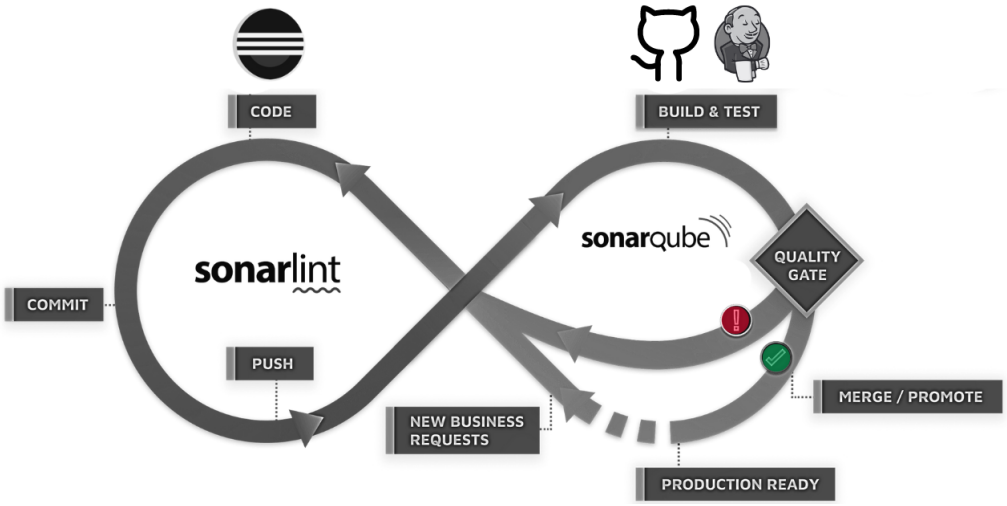


Figura 4.2: Etapas do Processo de Desenvolvimento com SonarQUBE no Contexto da Ativu Tecnologia.

Fonte: Adaptado de SonarQUBE [50]

Considerando que o SonarQube já estava previamente homologado e implantado na empresa, foi decidido realizar uma atualização da ferramenta. A versão instalada anteriormente estava defasada, datando de quatro anos atrás sendo a 7.7, e, portanto, foi atualizada para a versão mais recente, a 10.1. Através dessa atualização, a ferramenta foi enriquecida com melhorias de acurácia, visto que suas atualizações incorporaram aprimoramentos no tratamento de falsos positivos e falsos negativos. Esse refinamento contribuiu para uma validação mais sólida da eficiência da ferramenta, como é evidenciado na Figura 4.3.

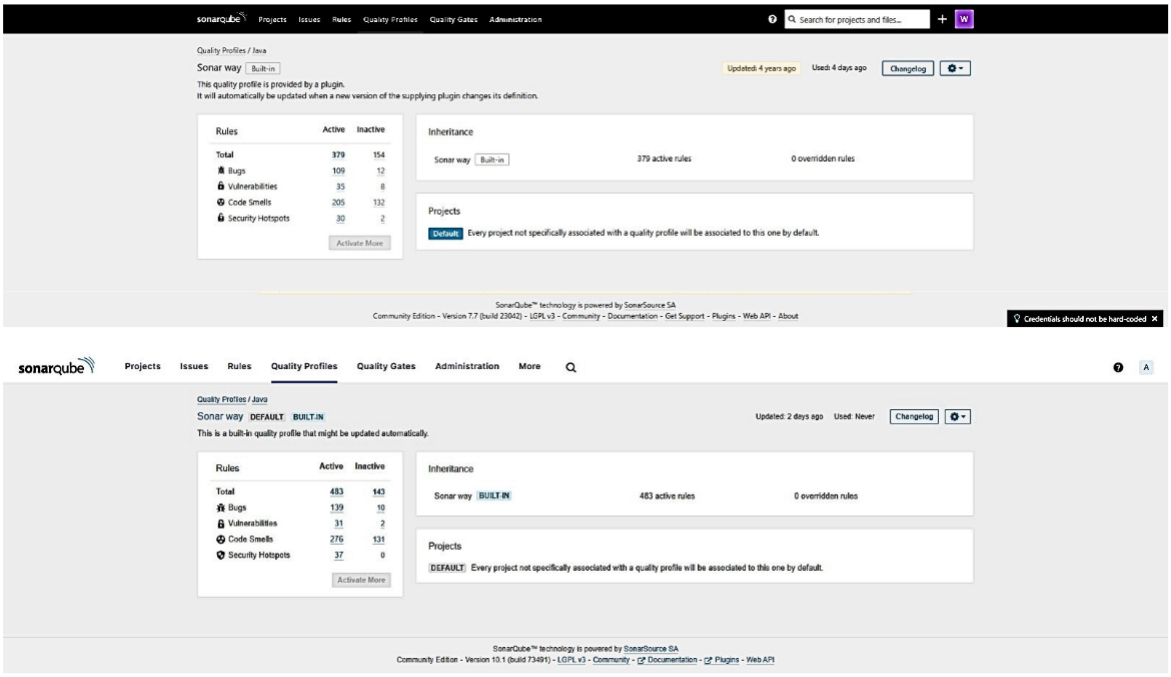


Figura 4.3: Atualização da Ferramenta SonarQUBE.
Fonte: Adaptado de SonarQUBE [50].

4.1.3 Elaboração do Relatório de Auditoria do Código Fonte - Ferramenta SonarQube

Com a finalidade de adquirir as informações necessárias, os *Scanners* foram executados, resultando na criação de um relatório, conforme exemplificado na Figura 4.4.

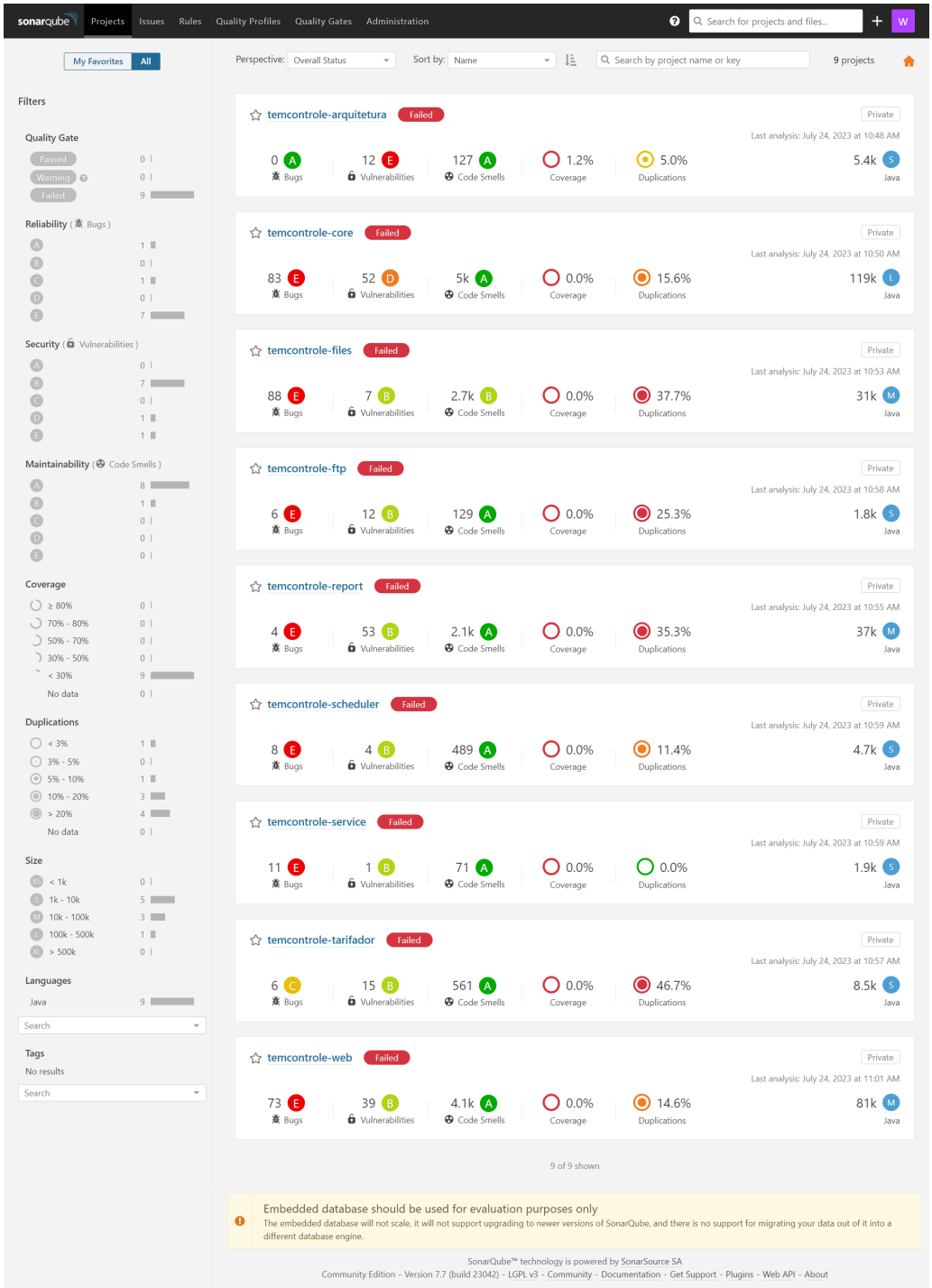


Figura 4.4: Relatório Ferramenta SonarQUBE - Auditoria do Código Fonte.
Fonte: Relatório de Resultados SonarQUBE [50].

4.1.4 Validação do Critério de Risco de Uso Indevido de Licença a Partir do Software FOSSology

Para validar o critério de risco relacionado ao uso indevido de licenças, foi utilizado a ferramenta FOSSology. Essa ferramenta está hospedada na mesma instância que contém o clone do repositório do código fonte do software AtivuControl no GitHub. A instância em questão é equipada com um processador de núcleo único e quatro threads, operando a 3,2 GHz, 16 GB de RAM e 1 TB de armazenamento. Durante a instalação, foi empregado o Dockerfile que permitiu a execução em contêineres, utilizando uma única instância em combinação com um banco de dados PostgreSQL externo. Isso permitiu criar tarefas específicas para cada módulo do sistema, responsáveis por executar as respectivas operações e coletar informações. Nesse contexto, as execuções dos módulos exigiram apenas alguns minutos, exceto pelo módulo Web, que levou algumas horas devido à sua maior quantidade de licenças.

4.1.5 Elaboração do Relatório de Auditoria do Código Fonte - Ferramenta FOSSOlogy

Com a finalidade de adquirir as informações necessárias, os jobs foram executados, resultando na criação de um relatório por módulo, conforme exemplificado nas Figuras 4.5, 4.6, 4.7, 4.8, 4.9 e 4.10.

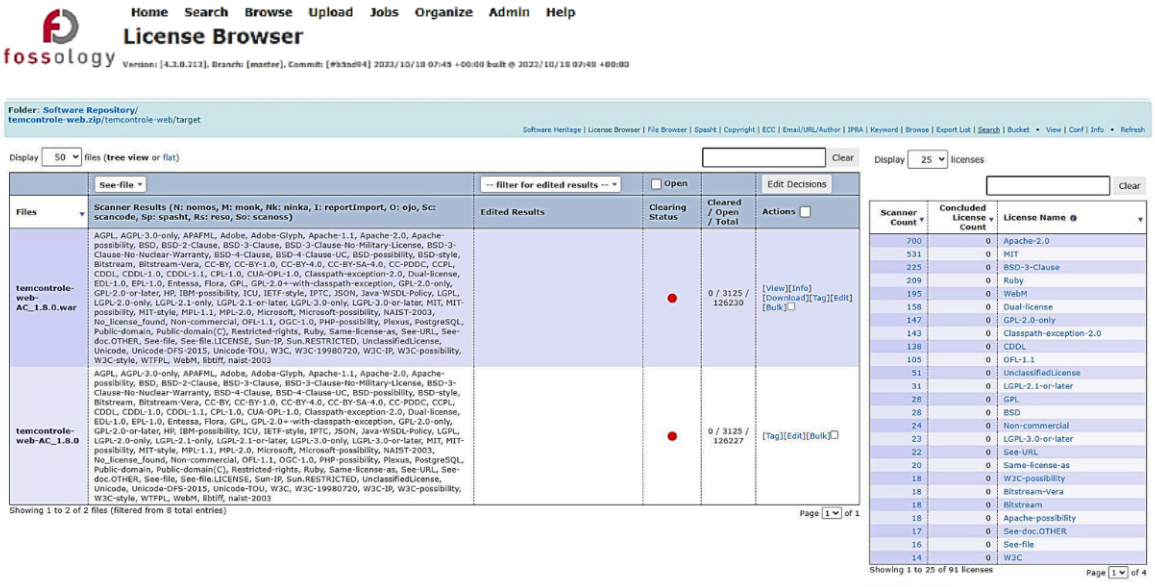


Figura 4.5: Relatório Ferramenta FOSSOlogy - Módulo Web.
Fonte: Relatório de Resultados FOSSOlogy [130].

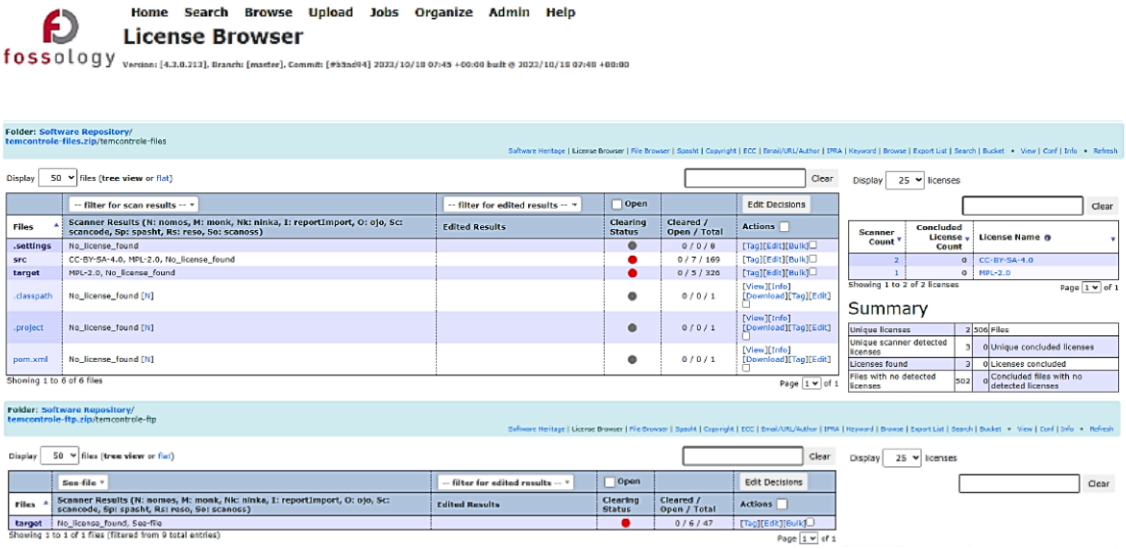


Figura 4.6: Relatório Ferramenta FOSSOlogy - Módulos Files e FTP.
Fonte: Relatório de Resultados FOSSOlogy [130].

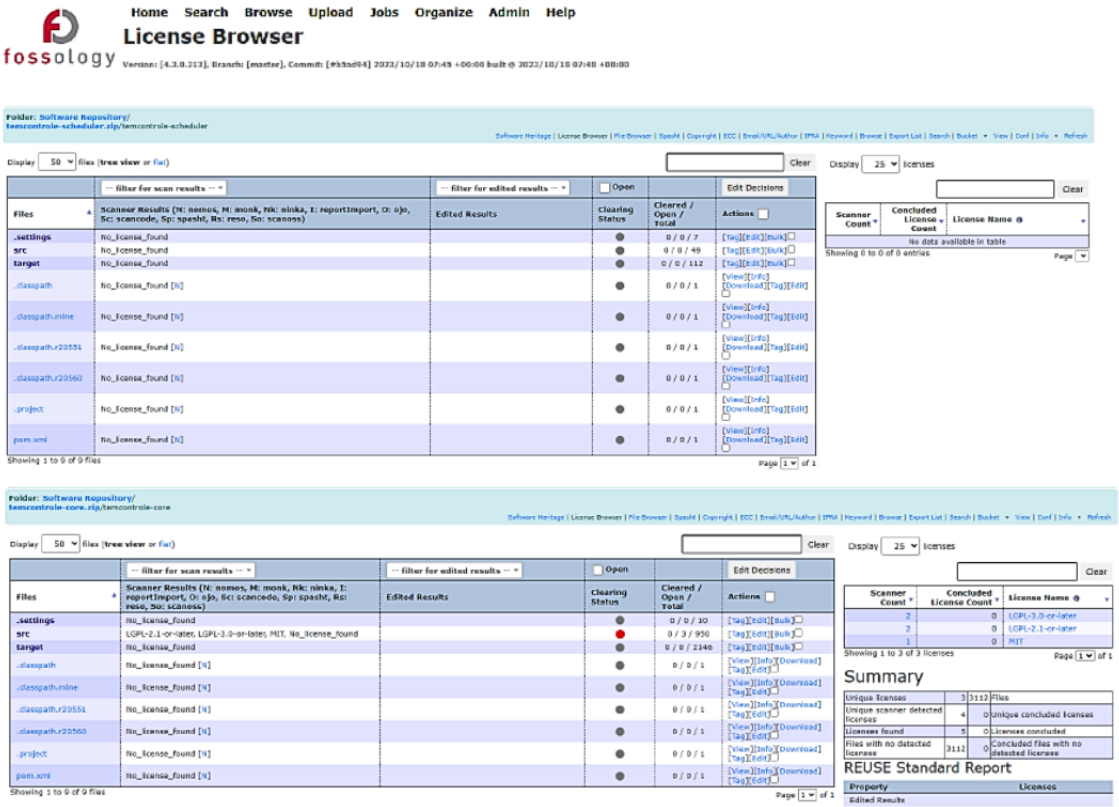


Figura 4.7: Relatório Ferramenta FOSSOlogy - Módulos Scheduler e Core.
Fonte: Relatório de Resultados FOSSOlogy [130].

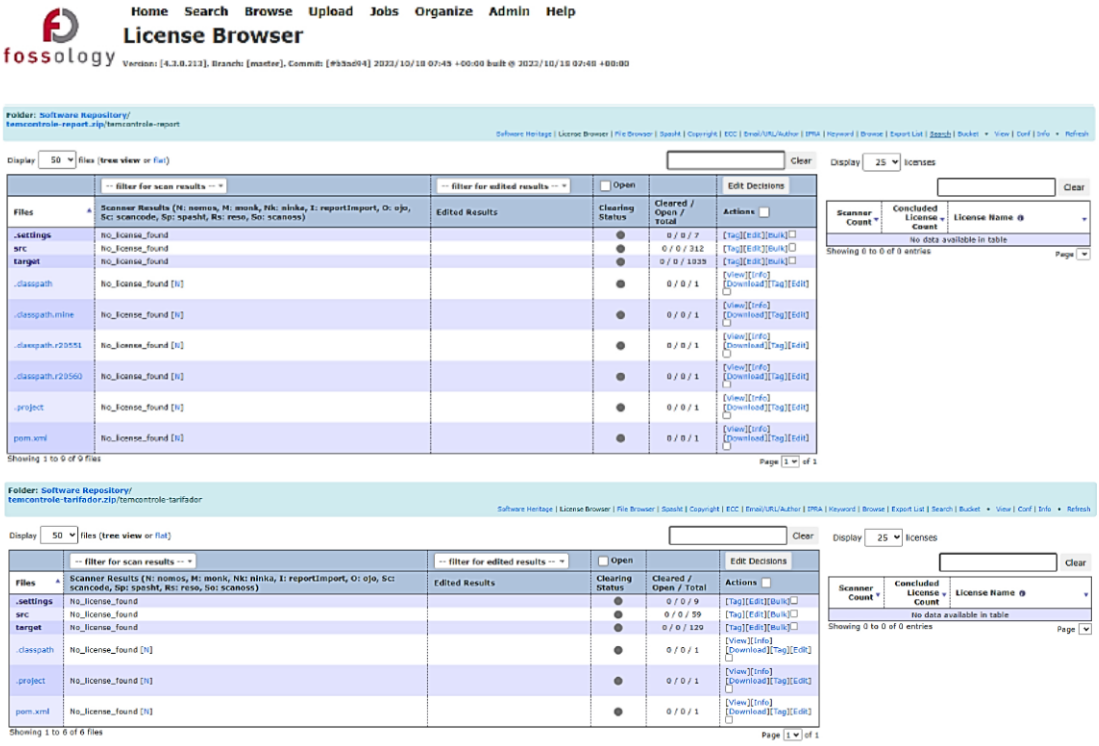


Figura 4.8: Relatório Ferramenta FOSSOlogy - Módulos Report e Tarifador.
Fonte: Relatório de Resultados FOSSOlogy [130].

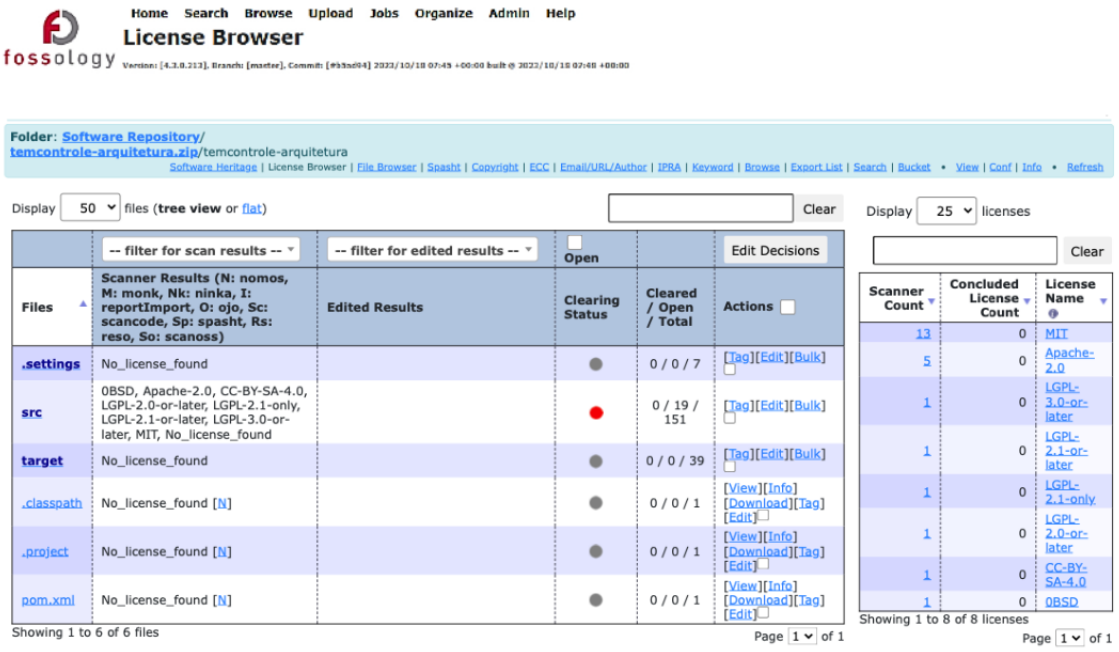


Figura 4.9: Relatório Ferramenta FOSSOlogy - Módulo Arquitetura.
Fonte: Relatório de Resultados FOSSOlogy [130].

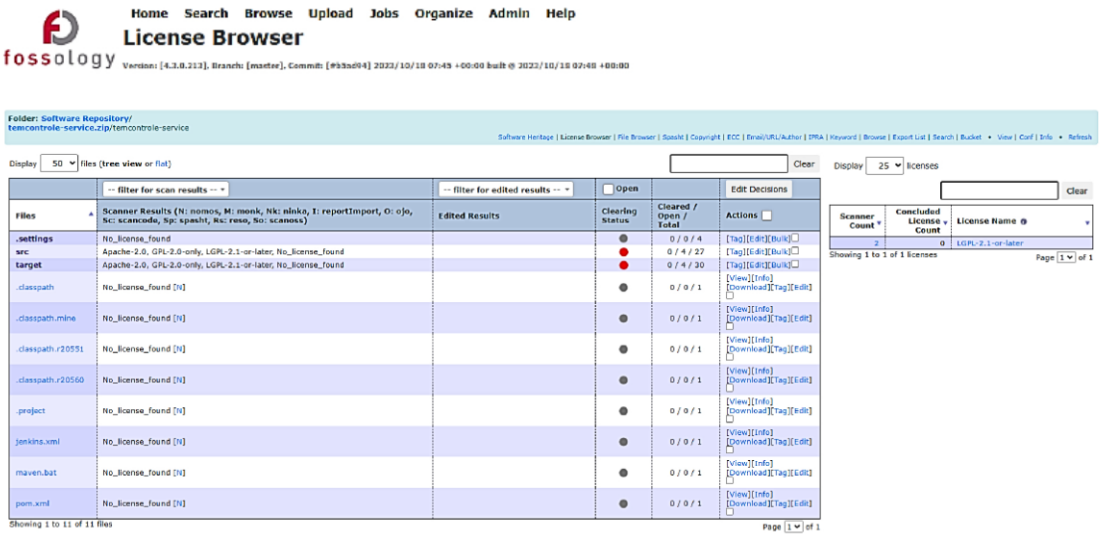


Figura 4.10: Relatório Ferramenta FOSSOlogy - Módulo Servece.
Fonte: Relatório de Resultados FOSSOlogy [130].

4.1.6 Validação do Critério de Risco de Plágio ou Similaridade a Partir do Sotfware CopySpider

Para validar o critério de risco relacionada ao uso de Plágio ou Similaridade, foi empregada a ferramenta CopySpider. O instalador do CopySpider possui 75 MB e requer uma versão atualizada do Java Runtime Environment. Essa ferramenta está hospedada na mesma instância que contém o clone do repositório do código fonte do software AtivuControl no GitHub. A instância em questão é equipada com um processador de núcleo único e quatro threads, operando a 3,2 GHz, 16 GB de RAM e 1 TB de armazenamento.

Durante a fase de configuração, dado que o CopySpider é uma ferramenta freeware para testar arquivos em busca de cópias indevidas na internet, foram selecionadas amostras de aproximadamente 50 arquivos por módulo, excluindo aqueles que contêm regras de negócio do software ATIVUControl por razões de segurança. Ao receber um arquivo de entrada para teste, o CopySpider analisou o arquivo inteiro, criando uma estrutura de dados com informações relevantes que foram utilizadas no processo de busca por arquivos semelhantes. Esta técnica de análise é chamada de *fingerprinting*.

Foi observado que quanto mais representativo for a *fingerprinting* de um arquivo, mais ágil e precisa será a busca por arquivos semelhantes, facilitando a identificação de cópias indevidas. Para isso, foi necessário utilizar uma conexão com a internet preferencialmente livre de bloqueios e limitadores de velocidade, pois isso afeta diretamente a velocidade de construção dos resultados. Isso permitiu a criação de tarefas específicas para cada módulo do sistema, responsáveis por executar as operações correspondentes e coletar informações.

Nesse contexto, as execuções dos módulos levaram, em média, 2 horas e 50 minutos, exceto quando não foi encontrado nenhum erro.

4.1.7 Elaboração do Relatório de Auditoria do Código Fonte - Ferramenta CopySpider

Com a finalidade de adquirir as informações necessárias, o programa foi executado, resultando na criação de um relatório por módulo, conforme exemplificado nas Figuras 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18 e 4.19.

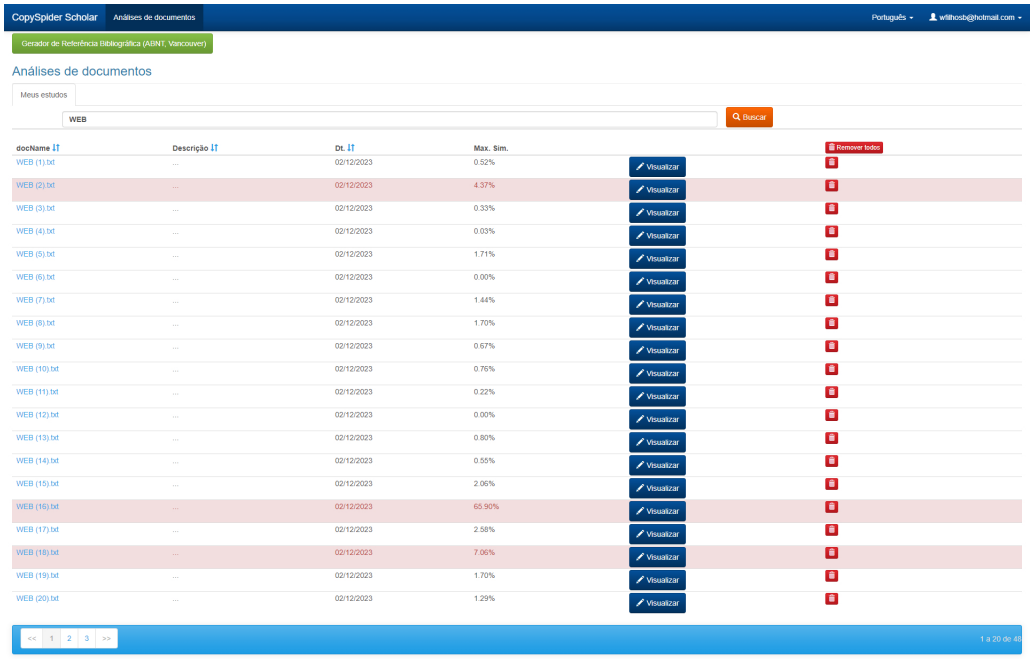


Figura 4.11: Relatório Ferramenta CopySpider - Módulo Web.
Fonte: Relatório de Resultados CopySpider [53].

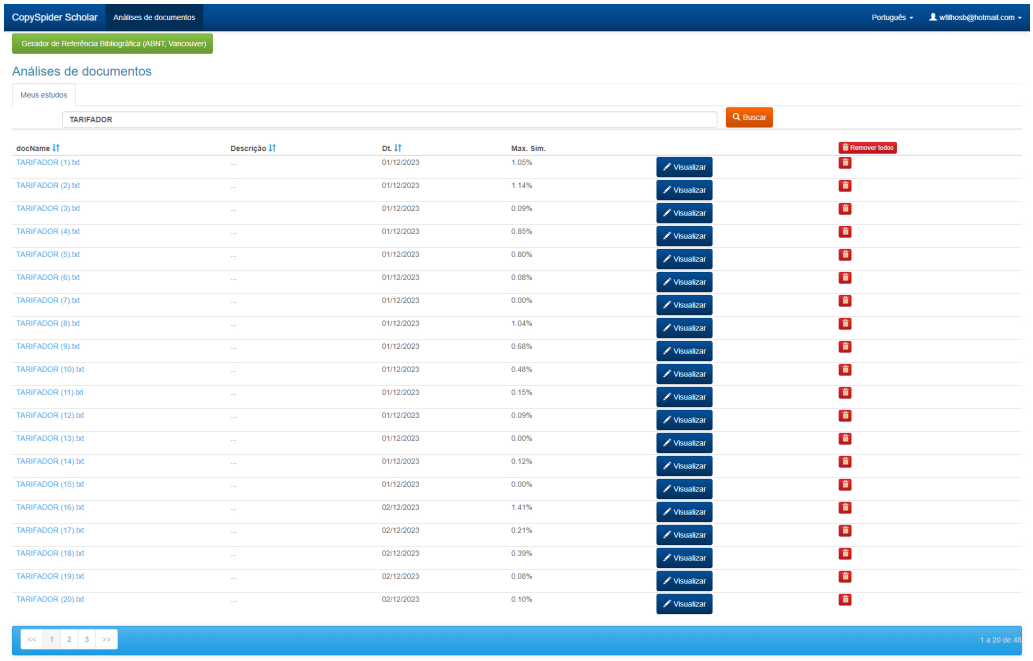


Figura 4.12: Relatório Ferramenta CopySpider - Módulo Tarifador.
Fonte: Relatório de Resultados CopySpider [53].

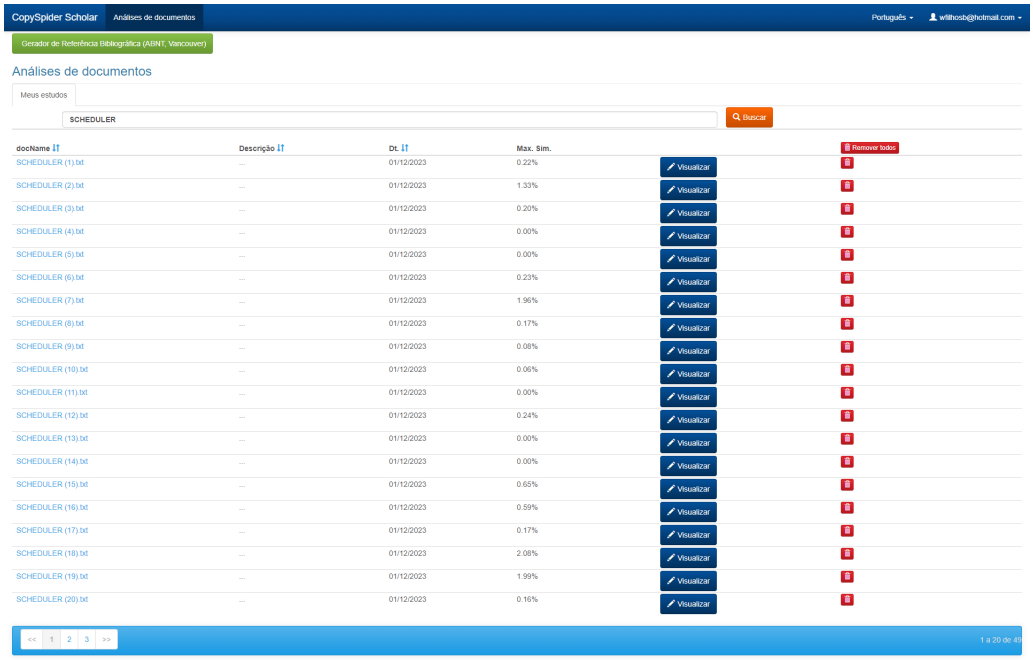


Figura 4.13: Relatório Ferramenta CopySpider - Módulo Scheduler.
Fonte: Relatório de Resultados CopySpider [53].

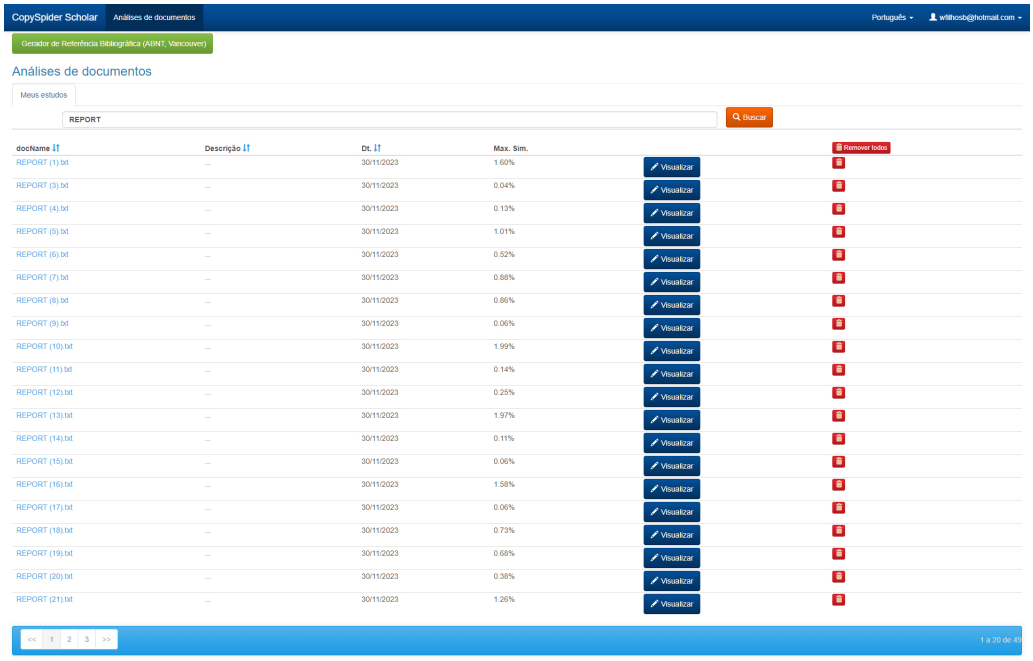


Figura 4.14: Relatório Ferramenta CopySpider - Módulo Report.
Fonte: Relatório de Resultados CopySpider [53].

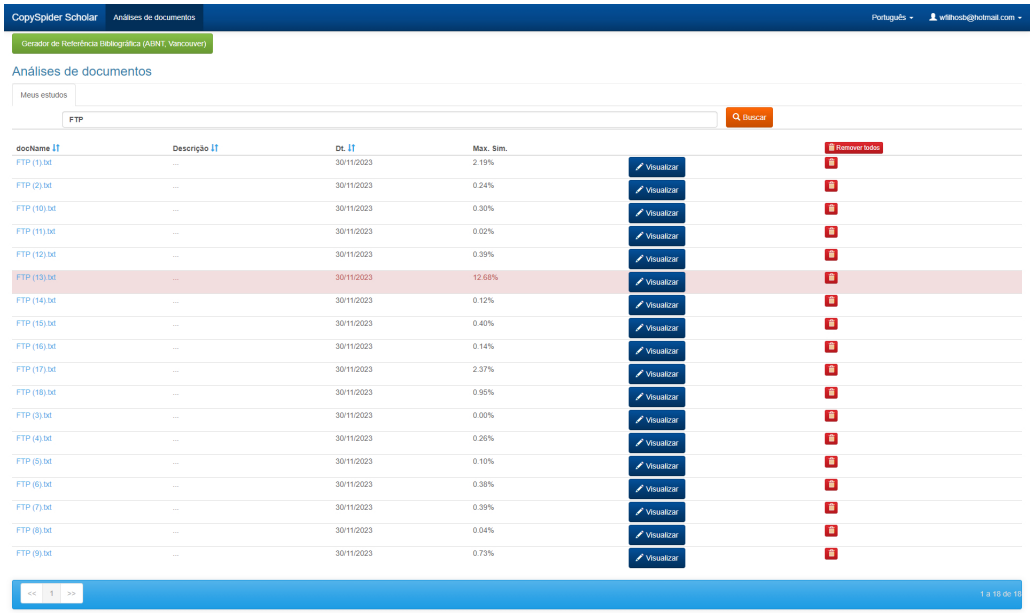


Figura 4.15: Relatório Ferramenta CopySpider - Módulo FTP.
Fonte: Relatório de Resultados CopySpider [53].

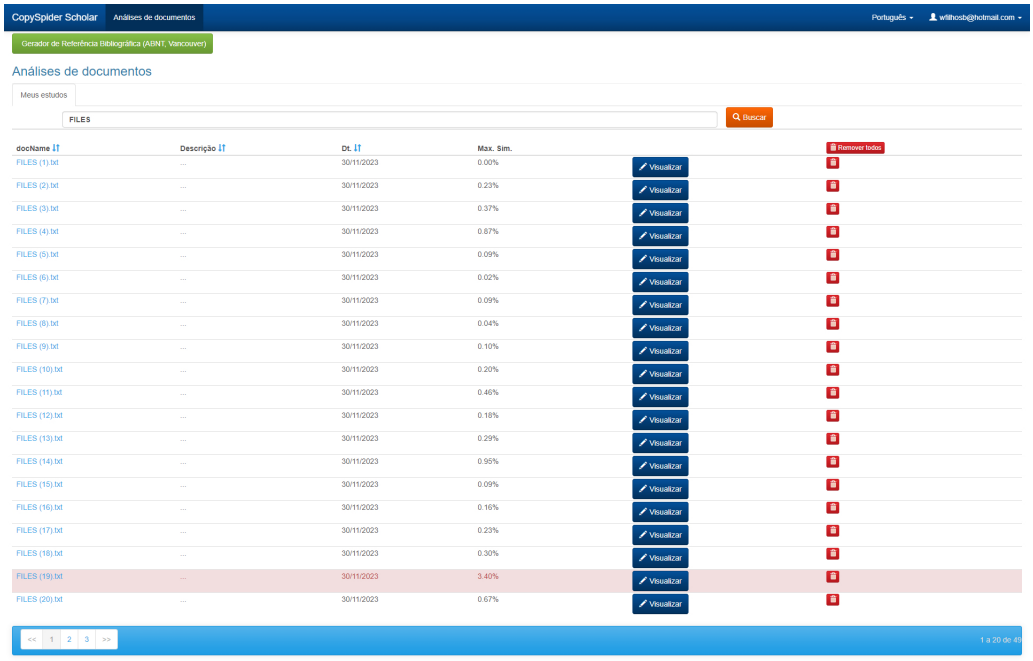


Figura 4.16: Relatório Ferramenta CopySpider - Módulo Files.
Fonte: Relatório de Resultados CopySpider [53].

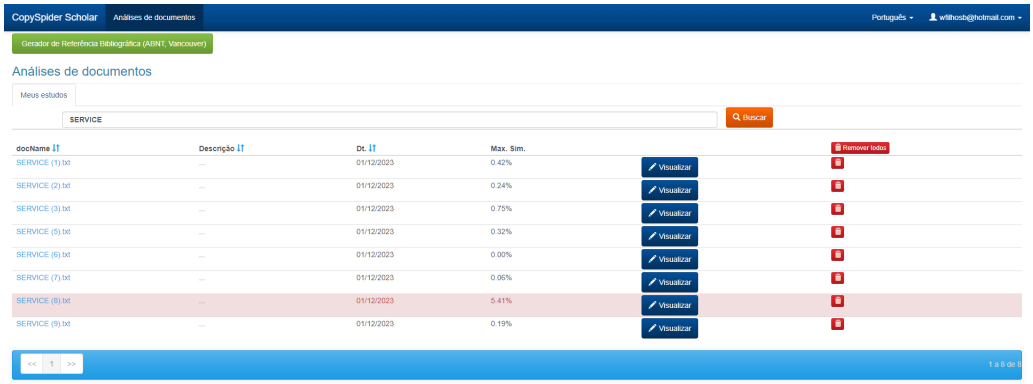


Figura 4.17: Relatório Ferramenta CopySpider - Módulo Service.
Fonte: Relatório de Resultados CopySpider [53].

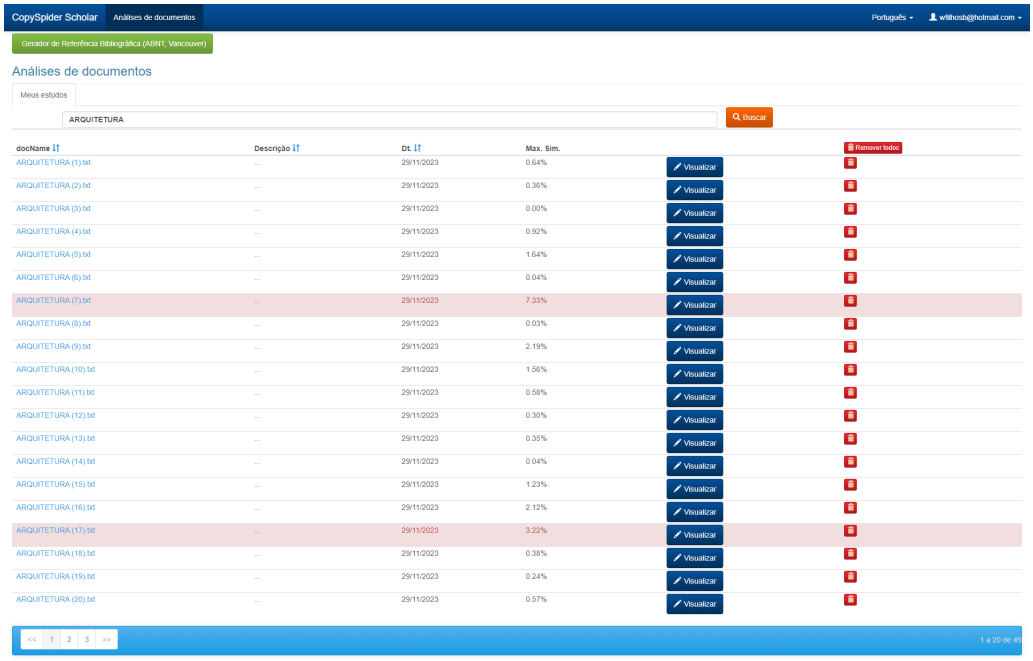


Figura 4.18: Relatório Ferramenta CopySpider - Módulo Arquitetura.
Fonte: Relatório de Resultados CopySpider [53].

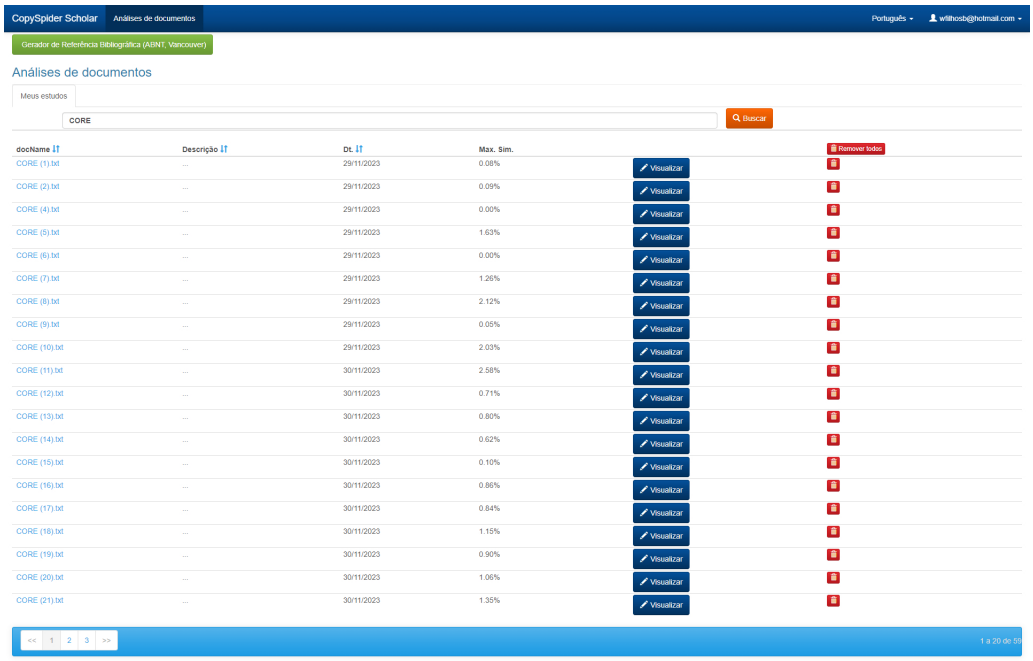


Figura 4.19: Relatório Ferramenta CopySpider - Módulo Core.
Fonte: Relatório de Resultados CopySpider [53].

Nos relatórios, gerados pelas ferramentas SonarQube, FOSSOlogy e CopySpider, além de validar os critérios de risco, foi possível identificar a existência de problemas no software que demandam atenção para evitar possíveis prejuízos à empresa Ativu Tecnologia.

Foram identificados problemas de vulnerabilidade na aplicação, assim como ocorrências de *bugs* e possíveis *Code Smells* que demandam melhorias no código fonte. Também foi possível identificar questões relacionadas as licenças que possuem uma versão comercial e necessitam de verificação quanto à conformidade atual. Também foi identificado uma porcentagem de similaridade ou possíveis casos de plágio em trechos do código fonte que necessita de tratamento. Como os problemas foram identificados nos nove módulos do software, tornou-se imprescindível estabelecer uma priorização com base na magnitude dos critérios de risco, determinando qual parte do software demanda tratamento imediato. Essa avaliação levou em consideração a complexidade de cada problema e os recursos disponíveis pela empresa. Essa abordagem estratégica permitiu direcionar os esforços de maneira eficiente, concentrando-se inicialmente nos aspectos mais críticos e impactantes.

4.2 Aplicação do Modelo MCDA

Com o intuito de abordar o problema decisório e elaborar o ranking de priorização levando em consideração os critérios de risco e as alternativas com os módulos do software, esta seção apresentará os seguintes passos:

1)Estruturação do problema, onde serão apresentados os critérios de risco e as alternativas a suas avaliações;

2)Construção do modelo, baseada na aplicação do método (FAHP), que permitirá uma análise precisa e ponderada;

3)Validação do modelo, com base na análise de consistência e sensibilidade, que fortalecerá a confiabilidade do modelo desenvolvido.

Dessa forma, busca assegurar uma abordagem sólida e eficaz para o processo de priorização, facilitando a tomada de decisão e os resultados.

4.2.1 Estruturação do Problema Decisório

A fase de Estruturação do Problema Decisório consiste em compreender e identificar o problema de decisão, buscando sua devida estruturação. Isso inclui a identificação das possíveis soluções/alternativas e a análise de sua viabilidade, determinadas ao contexto a ser investigado [133].

O problema decisório deste trabalho reside na criação de um *ranking* de priorização que ordene as alternativas de acordo com o risco de forma decrescente em relação aos módulos do sistema, considerando os critérios de risco identificados.

4.2.2 Definição dos Critérios de Risco e das Alternativas

Os critérios de risco foram identificados através da literatura e na auditoria do código fonte realizada no software ATIVUControl, foram validados, sendo: Vulnerabilidade, *Bug*, *Code Smells*, Plágio ou Similaridade e uso indevido de licença. Com isso, posteriormente os critérios de risco foram discutidos e reconhecidos através de um brainstorming com os especialistas da empresa, conforme sugerido por Saaty [134]. Os critérios de risco estão ilustrados na tabela 4.3.

Tabela 4.3: Critério de Risco.

CR	**Critério de Risco**
CR1	Vulnerabilidade
CR2	*Code Smells*
CR3	*Bugs*
CR4	Plágio/similaridade
CR5	Licença de software indevida

Devido à robustez do software ATIVUControl e à interconexão dos seus módulos, a tarefa de escolher qual módulo deve ser priorizada torna-se complexa mesmo após a aplicação da auditoria no código fonte e obtenção dos resultados. Para facilitar essa escolha, as alternativas foram identificadas pelos nove módulos que compõem o software, conforme ilustrado na tabela 4.4. Essa abordagem permite uma análise mais específica e direcionada, permitindo a identificação dos módulos mais críticos em termos de risco e priorização das ações de forma mais assertiva.

Tabela 4.4: Módulo software ATIVUControl.

MS	Módulo do Software
M1	Arquitetura
M2	Core
M3	Files
M4	FTP
M5	Report
M6	Scheduler
M7	Service
M8	Tarifador
M9	Web

4.2.3 Decomposição Hierárquica do Problema

Em um modelo AHP geral, o objetivo é representado no primeiro nível, seguido pelos critérios e subcritérios no segundo e terceiro níveis, respectivamente. Por fim, as alternativas são oferecidas no quarto nível [76]. Dessa forma, o problema decisório está organizado da seguinte maneira:

- Primeiro nível: Ranking de priorização de risco dos módulos do software ATIVUControl;
- Segundo e terceiro níveis: Critérios de risco identificados na Revisão Sistemática da Literatura (RSL) e validados no software, conforme ilustrado na tabela 4.3;
- Quarto nível: Alterativas complexas dos módulos do sistema ATIVUControl, conforme ilustrado na tabela 4.4.

Para aprimorar a compreensão do modelo decisório e alcançar o objetivo, foi desenvolvida uma hierarquia estruturada, conforme representado na figura 4.20.

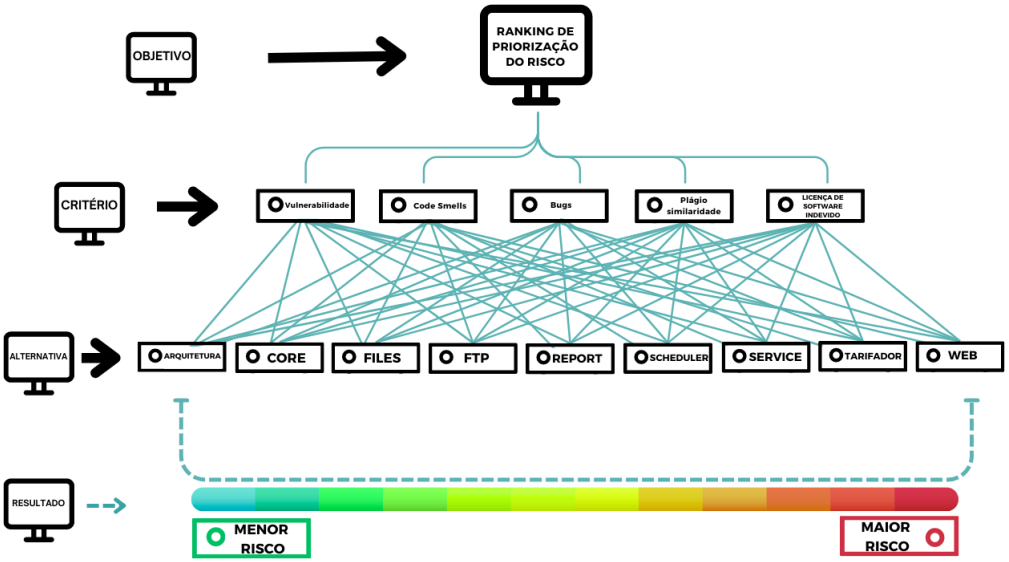


Figura 4.20: Decomposição Hierárquica do Problema.
Fonte: Adaptada de Buckley [74].

4.3 Construção do Modelo pelo FAHP

4.3.1 Determinando os Pesos dos Critérios

Para estabelecer os critérios e avaliar as alternativas no processo de priorização do risco, conduziu-se uma reunião estruturada com os funcionários mais capacitados do time de desenvolvimento da empresa Ativu, composta por seis membros, incluindo gerente, especialista e analista, conforme apresentado na tabela 4.5.

Tabela 4.5: Informações dos especialistas.

CARGO	**FORMAÇÃO**	**TEMPO DE EXPERIÊNCIA**
Gerente de Desenvolvimento	Bacharelado em ciência da computação	16 anos
Especialista de Desenvolvimento	Pós Graduado em Arquitetura de Software	12 anos
Especialista de Teste	Bacharelado em sistema de informação	8 anos
Gerente de Requisitos	Bacharelado em ciência da computação	8 anos
Analista de desenvolvimento	Bacharelado em sistema de informação	3 anos
Analista de desenvolvimento	Bacharelado em ciência da computação	7 anos

Para determinar as preferências, adotou-se o método Planning Poker, já empregado na empresa e no qual a equipe de desenvolvimento possui proficiência. Este método, introduzido por James Grenning [135], combina aspectos de estimativas comparativas e técnicas de tomada de decisão em grupo, como a técnica Delphi [136]. Além disso, ele incorpora uma sequência de Fibonacci que facilita as estimativas. A representação da comparação média par a par dos critérios foram alinhadas com as preferências da equipe durante esse processo, conforme ilustrada na Tabela 4.6.

Tabela 4.6: Matriz de comparação para critérios.

CR	**R1**	**R2**	**R3**	**R4**	**R5**
CR1	(1,1,1)	(6,7,8)	(2,3,4)	(9,9,9)	(4,5,6)
CR2	(1/8,1/7,1/6)	(1,1,1)	(1/6,1/5,1/4)	(2,3,4)	(1/4,1/3,1/2)
CR3	(1/4,1/3,1/2)	(4,5,6)	(1,1,1)	(6,7,8)	(2,3,4)
CR4	(1/9,1/9,1/9)	(1/4,1/3,1/2)	(1/8,1/7,1/6)	(1,1,1)	(1/6,1/5,1/4)
CR5	(1/6,1/5,1/4)	(2,3,4)	(1/4,1/3,1/2)	(4,5,6)	(1,1,1)

Para auxiliar no cálculo, foi empregada a biblioteca Python (pyDecision.algorithm), especialmente com o módulo (fuzzy AHP method)[132]. Esse módulo, ao receber os dados resultantes das comparações par a par, realizou os cálculos por meio das funções disponíveis nesse pacote. Dessa forma, as médias geométricas dos valores de comparação fuzzy de todos os critérios foram obtidas. Além disso, os valores totais e seus inversos

também foram calculados e exibidos. Uma vez que os números triangulares fuzzy devem estar em ordem crescente, a sequência dos números foi ajustada. Consequentemente, o peso relativo não fuzzy de cada critério foi calculado por meio da média dos números fuzzy correspondentes. Utilizando os valores não fuzzy, os pesos normalizados de cada critério foram calculados e organizados, conforme demonstrado na Tabela 4.7.

Tabela 4.7: Pesos Relativos Médios e Normalizados dos Critérios.

CR	**Pesos Fuzzy**	**Média dos Pesos**	**Pesos Normalizados**
CR1	(0,371 0,51 0,69)	0,523	**0,503**
CR2	(0,044 0,064 0,094)	0,067	**0,065**
CR3	(0,181 0,264 0,387)	0,277	**0,267**
CR4	(0,025 0,033 0,046)	0,035	**0,033**
CR5	(0,088 0,13 0,193)	0,137	**0,132**

4.3.2 Determinando os Pesos das Alternativas em Relação aos Critérios

Após obter os pesos relativos não fuzzy normalizados para os critérios, a metodologia foi reaplicada para determinar os valores correspondentes das alternativas. No entanto, neste caso, as alternativas foram avaliadas em pares em relação a cada critério específico. Ou seja, esse processo foi repetido cinco vezes, uma para cada critério, como demonstrado nas Figuras 4.21,4.22.

CR1	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/8, 1/7, 1/6)	(1/8, 1/7, 1/6)	(1/9, 1/9, 1/9)
M2	(6,7,8)	(1, 1, 1)	(4,5,6)	(4,5,6)	(4,5,6)	(6,7,8)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M3	(4,5,6)	(1/6, 1/5, 1/4)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/6, 1/5, 1/4)
M4	(4,5,6)	(1/6, 1/5, 1/4)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/6, 1/5, 1/4)
M5	(2,3,4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)
M6	(2,3,4)	(1/8, 1/7, 1/6)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/9, 1/9, 1/9)
M7	(6,7,8)	(1, 1, 1)	(2,3,4)	(2,3,4)	(4,5,6)	(4,5,6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M8	(6,7,8)	(1, 1, 1)	(2,3,4)	(2,3,4)	(4,5,6)	(4,5,6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M9	(9,9,9)	(2,3,4)	(4,5,6)	(4,5,6)	(6,7,8)	(9,9,9)	(2,3,4)	(2,3,4)	(1, 1, 1)

CR2	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/9, 1/9, 1/9)
M2	(6,7,8)	(1, 1, 1)	(1, 1, 1)	(4,5,6)	(6,7,8)	(4,5,6)	(9,9,9)	(2,3,4)	(1, 1, 1)
M3	(6,7,8)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(4,5,6)	(4,5,6)	(6,7,8)	(1, 1, 1)	(1/4, 1/3, 1/2)
M4	(4,5,6)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(2,3,4)	(2,3,4)	(6,7,8)	(1/4, 1/3, 1/2)	(1/6, 1/5, 1/4)
M5	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)
M6	(2,3,4)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)
M7	(1, 1, 1)	(1/9, 1/9, 1/9)	(1/8, 1/7, 1/6)	(1/8, 1/7, 1/6)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/9, 1/9, 1/9)
M8	(6,7,8)	(1/4, 1/3, 1/2)	(1, 1, 1)	(2,3,4)	(4,5,6)	(4,5,6)	(6,7,8)	(1, 1, 1)	(1/4, 1/3, 1/2)
M9	(9,9,9)	(1, 1, 1)	(2,3,4)	(4,5,6)	(6,7,8)	(6,7,8)	(9,9,9)	(2,3,4)	(1, 1, 1)

CR3	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/9, 1/9, 1/9)
M2	(6,7,8)	(1, 1, 1)	(2,3,4)	(4,5,6)	(6,7,8)	(6,7,8)	(6,7,8)	(2,3,4)	(1/4, 1/3, 1/2)
M3	(4,5,6)	(1/4, 1/3, 1/2)	(1, 1, 1)	(2,3,4)	(4,5,6)	(4,5,6)	(6,7,8)	(1, 1, 1)	(1/4, 1/3, 1/2)
M4	(4,5,6)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(2,3,4)	(1, 1, 1)	(6,7,8)	(1/4, 1/3, 1/2)	(1/6, 1/5, 1/4)
M5	(2,3,4)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)
M6	(2,3,4)	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)
M7	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/8, 1/7, 1/6)	(1/8, 1/7, 1/6)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/8, 1/7, 1/6)	(1/9, 1/9, 1/9)
M8	(6,7,8)	(1/4, 1/3, 1/2)	(1, 1, 1)	(2,3,4)	(4,5,6)	(4,5,6)	(6,7,8)	(1, 1, 1)	(1/4, 1/3, 1/2)
M9	(9,9,9)	(2,3,4)	(2,3,4)	(4,5,6)	(6,7,8)	(6,7,8)	(9,9,9)	(2,3,4)	(1, 1, 1)

Figura 4.21: Matriz de Comparação das Alternativas em Relação aos Critérios R1, R2 e R3.

Fonte: Elaboração própria.

CR4	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	(1, 1, 1)	(1/8, 1/7, 1/6)	(1, 1, 1)	(1, 1, 1)	(4,5,6)	(4,5,6)	(1, 1, 1)	(2,3,4)	(1, 1, 1)
M2	(6,7,8)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(4,5,6)	(4,5,6)	(1, 1, 1)	(2,3,4)	(1, 1, 1)
M3	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(4,5,6)	(4,5,6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M4	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M5	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)
M6	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(2,3,4)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1/6, 1/5, 1/4)
M7	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(1, 1, 1)	(1, 1, 1)	(1, 1, 1)
M8	(1/4, 1/3, 1/2)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)
M9	(1, 1, 1)	(1, 1, 1)	(2,3,4)	(2,3,4)	(2,3,4)	(4,5,6)	(1, 1, 1)	(2,3,4)	(1, 1, 1)

CR5	M1	M2	M3	M4	M5	M6	M7	M8	M9
M1	(1, 1, 1)	(4,5,6)	(4,5,6)	(2,3,4)	(6,7,8)	(6,7,8)	(4,5,6)	(6,7,8)	(1/4, 1/3, 1/2)
M2	(1/6, 1/5, 1/4)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)	(4,5,6)	(4,5,6)	(2,3,4)	(4,5,6)	(1/8, 1/7, 1/6)
M3	(1/6, 1/5, 1/4)	(1, 1, 1)	(1, 1, 1)	(1/6, 1/5, 1/4)	(4,5,6)	(4,5,6)	(1, 1, 1)	(4,5,6)	(1/8, 1/7, 1/6)
M4	(1/4, 1/3, 1/2)	(2,3,4)	(4,5,6)	(1, 1, 1)	(6,7,8)	(6,7,8)	(4,5,6)	(6,7,8)	(1/6, 1/5, 1/4)
M5	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/9, 1/9, 1/9)
M6	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/9, 1/9, 1/9)
M7	(1/6, 1/5, 1/4)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/6, 1/5, 1/4)	(2,3,4)	(2,3,4)	(1, 1, 1)	(2,3,4)	(1/8, 1/7, 1/6)
M8	(1/8, 1/7, 1/6)	(1/6, 1/5, 1/4)	(1/6, 1/5, 1/4)	(1/8, 1/7, 1/6)	(1, 1, 1)	(1, 1, 1)	(1/4, 1/3, 1/2)	(1, 1, 1)	(1/9, 1/9, 1/9)
M9	(2,3,4)	(6,7,8)	(6,7,8)	(4,5,6)	(9,9,9)	(9,9,9)	(6,7,8)	(9,9,9)	(1, 1, 1)

Figura 4.22: Matriz de Comparação das Alternativas em Relação aos Critérios R4 e R5.

Fonte: Elaboração própria.

De maneira análoga à metodologia de cálculo dos critérios e as médias geométricas dos valores de comparação fuzzy são registradas para cada alternativa, considerando cada critério. Com base nessas explanações, os pesos relativos não fuzzy normalizados de cada alternativa para cada critério são identificados e registrados na Tabela 4.8.

Tabela 4.8: Pesos relativos não-fuzzy normalizados de cada alternativa para cada critério.

ID	**PCR**	**PM1**	**PM2**	**PM3**	**PM4**	**PM5**	**PM6**	**PM7**	**PM8**	**PM9**
CR1	0,503	0,017	0,176	0,065	0,065	0,032	0,003	0,152	0,152	0,312
CR2	0,065	0,022	0,229	0,161	0,077	0,033	0,039	0,018	0,144	0,276
CR3	0,267	0,002	0,231	0,136	0,067	0,037	0,041	0,019	0,141	0,307
CR4	0,033	0,123	0,189	0,012	0,107	0,032	0,038	0,012	0,086	0,184
CR5	0,132	0,224	0,079	0,065	0,157	0,022	0,022	0,050	0,022	0,361

A Tabela 4.9 apresenta a distribuição dos pesos dos critérios para cada alternativa. Na penúltima linha inferior, temos a agregação com os pesos ponderados totais do critério, e na última linha, os valores estão organizados para posterior transposição e classificação do ranking.

Tabela 4.9: Resultados agregados para cada alternativa de acordo com cada critério.

CR	**PM1**	**PM2**	**PM3**	**PM4**	**PM5**	**PM6**	**PM7**	**PM8**	**PM9**
CR1	0,009	0,089	0,033	0,033	0,016	0,002	0,076	0,076	0,157
CR2	0,001	0,015	0,010	0,005	0,002	0,003	0,001	0,009	0,018
CR3	0,001	0,062	0,036	0,018	0,010	0,011	0,005	0,038	0,082
CR4	0,004	0,006	0,000	0,004	0,001	0,001	0,000	0,003	0,006
CR5	0,030	0,010	0,009	0,021	0,003	0,003	0,007	0,003	0,048
Total	**0,044**	**0,182**	**0,088**	**0,080**	**0,032**	**0,019**	**0,090**	**0,129**	**0,311**
Ordem	**7**	**2**	**5**	**6**	**8**	**9**	**4**	**3**	**1**

Os dados avaliados pelo FAHP foram ordenados por ordem de preferência de acordo com os valores apresentados na Figura 4.23. O ranking foi estabelecido em ordem decrescente do coeficiente de proximidade com a alternativa que possui o maior risco.

Ranking	MS	Módulo	Pontuação	Percentual
1	M9	Web	0,331	34,57%
2	M2	Core	0,182	19,01%
3	M8	Tarifador	0,129	13,47%
4	M7	Service	0,090	9,38%
5	M3	Files	0,088	9,19%
6	M4	FTP	0,080	8,36%
7	M1	Arquitetura	0,044	4,60%
8	M5	Report	0,032	3,34%
9	M6	Scheduler	0,019	1,98%

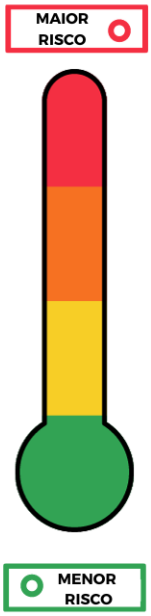


Figura 4.23: Ranking e Resultados da Aplicação do FAHP.
Fonte: Elaboração própria.

4.4 Validação do Modelo

Segundo Qureshi, Harrison e Wegener [137], três abordagens garantem a adequação de um modelo MCDA (Múltiplos Critérios de Decisão) desenvolvido de acordo com a metodologia aplicada:

- Verificação;
- Análise de Sensibilidade;
- Validação.

Neste contexto, para validar a verificação, este trabalho adotou o Índice de Consistência Relativa (ICR), seguindo os cálculos sugeridos por Saaty [58]. O resultado dessa verificação indica uma consistência satisfatória nas seis matrizes criadas com base na técnica do Planning Poker para coletar as contribuições dos especialistas. Essa consistência está dentro do limite de até 10%, conforme considerado para esse índice. A seguir, os resultados são apresentados de forma detalhada na Figura 4.24.

Peso Critério de Risco	ICR	Stutus
CR1	6%	OK
CR2		
CR3		
CR4		
CR5		

Peso Bugs	ICR	Stutus
M1	6%	OK
M2		
M3		
M4		
M5		
M6		
M7		
M8		
M9		

Peso Vulnerabilidade	ICR	Stutus
M1	4%	OK
M2		
M3		
M4		
M5		
M6		
M7		
M8		
M9		

Peso Plágio/Similaridade	ICR	Stutus
M1	8%	OK
M2		
M3		
M4		
M5		
M6		
M7		
M8		
M9		

Peso Code Smell	ICR	Stutus
M1	5%	OK
M2		
M3		
M4		
M5		
M6		
M7		
M8		
M9		

Peso Licença Indevida	ICR	Stutus
M1	7%	OK
M2		
M3		
M4		
M5		
M6		
M7		
M8		
M9		

Figura 4.24: Matriz de Validação do índice de Consistência Relativa.
Fonte: Elaboração própria.

Para aprofundar ainda mais a validação do modelo, ele foi apresentado aos especialistas, gerentes e analistas de desenvolvimento da empresa ATIVU que participaram desse estudo. Foi notável o alto engajamento da equipe de desenvolvimento no projeto, evidenciado por discussões substanciais e ponderações cuidadosas durante a seleção dos critérios iniciais. Essas deliberações foram cruciais para definir as prioridades entre os cenários de riscos e suas alternativas.

Os participantes reconheceram o modelo como sólido e alinhado com a realidade, percebendo seu potencial para auxiliar o time de desenvolvimento na priorização estratégica das ações de mitigação de riscos identificadas na auditoria do código fonte do software. Isso fortalece a confiança na utilidade e eficácia do modelo proposto.

4.4.1 Análise de Sensibilidade

A exploração da análise de sensibilidade, é uma abordagem que pode ser aplicada individualmente ou em conjunto, é empregada para avaliar a estabilidade e coesão de um

modelo quando ocorrem modificações em seus parâmetros [137], [56]. Segundo Triantaphyllou (1997) [138], essa exploração é crucial para compreender como a ordenação das bases de dados pode oscilar à medida que os pesos dos critérios são ajustados. Conforme o peso de um critério é incrementado, a relevância dos demais critérios é proporcionalmente reduzida, resultando no recálculo das prioridades gerais das opções em análise.

Antes da validação dos critérios de risco relacionados a Plágio ou Similaridade e ao Uso Indevido de Licença, foi realizada uma primeira rodada de discussão com os especialistas para preencher a matriz FAHP, na qual foram adquiridos os pesos e obtido os resultados preliminares. Após a validação dos critérios de plágio ou similaridade e ao uso indevido de licença no software, foram identificados os problemas. Com o auxílio do modelo criado usando o método Planning Poker[135] e a biblioteca Python (pyDecision.algorithm)[132], especificamente o módulo FAHP foi realizado uma segunda rodada de validação de pesos com os especialistas.

Após ajustes nos parâmetros, como critérios e alternativas, é notável uma leve alteração na ordem dos pesos entre os critérios de risco *Code Smell* e Uso Indevido de Licença. Também houve uma sutil modificação na ordem dos pesos das alternativas, notadamente entre os módulos Service e Files, e entre os módulos Arquitetura e Report. Isso confirma que o modelo é capaz de gerar dois rankings com resultados distintos, demonstrando sua adaptabilidade e a capacidade de fornecer resultados variados. Nas Figuras 4.25,4.26, 4.27 e 4.28, são apresentados uma comparação visual desses resultados ordenados.

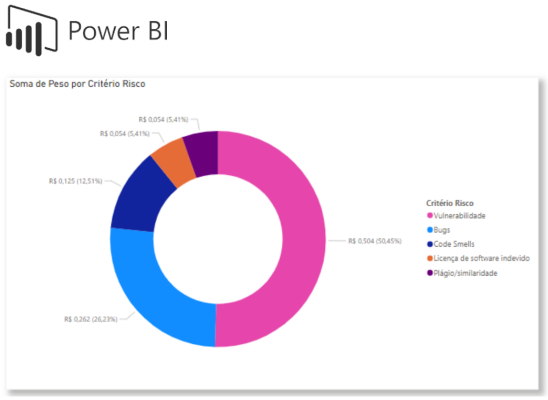


Figura 4.25: Ranking Primeira Rodada (CR)

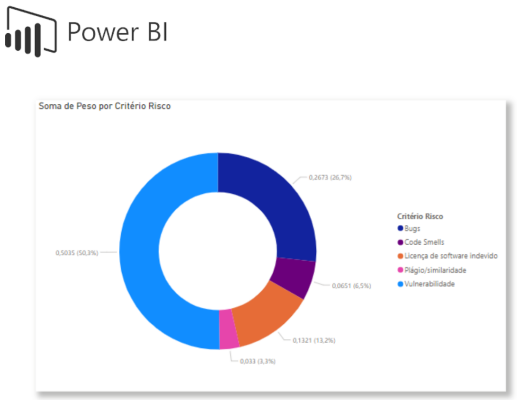


Figura 4.26: Ranking Segunda Rodada (CR)

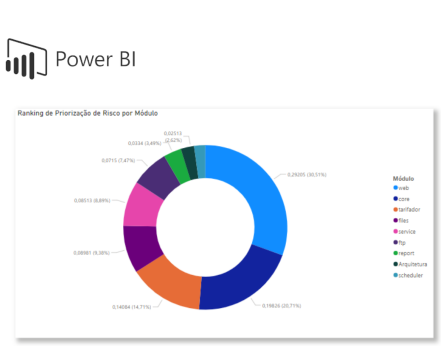


Figura 4.27: Ranking Primeira Rodada(MS)

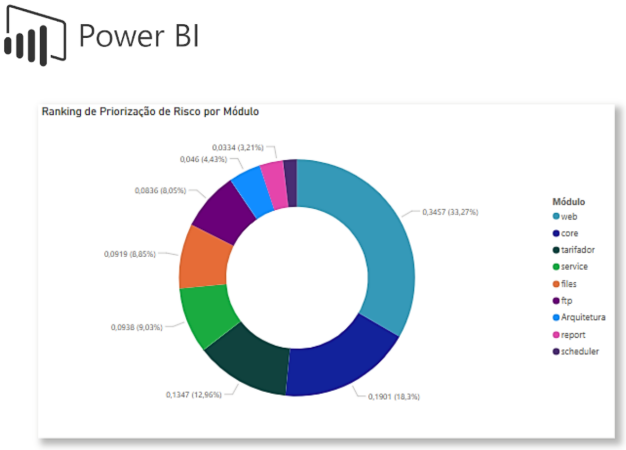


Figura 4.28: Ranking Segunda Rodada(MS)

4.5 Conclusão Parcial

Com base nos *insights* provenientes da Literatura, que permitiu a identificação dos critérios, métodos e ferramentas para a auditoria de código fonte, o estudo de caso foi conduzido para construir o modelo proposto nesta pesquisa. Esse modelo foi fundamentado nas etapas de execução do software de auditoria, as quais validaram os critérios de risco e a problemática no software, em conjunto com a aplicação do método MCDA. A utilização da ferramenta de auditoria surgiu como um desafio, visando a confirmação dos critérios de risco e a identificação de possíveis problemas no software.

Dessa forma, foi introduzido um relatório de auditoria, revelando as questões identificadas no software. Com a realização das etapas do processo MCDA, incluindo a Estruturação do Problema e a Definição dos Critérios de Risco, bem como a Decomposição dos elementos essenciais para a ordenação das alternativas com base na priorização do risco, foi possível aplicar os métodos multicritério FAHP.

Dando continuidade à Fase do Estudo de Caso, foram apresentados os resultados do Índice de Consistência Relativo para as seis matrizes construídas, acompanhado dos comentários fornecidos pelos especialistas, gerentes e analistas do setor de desenvolvimento. Esses contribuíram para o processo de avaliação e aprimoramento do modelo desenvolvido. No próximo capítulo, serão apresentados os resultados e análises.

Capítulo 5

Resultados e Análises

Neste capítulo, são apresentados os resultados e as análises obtidas. Com o intuito de aprimorar a compreensão e o processo de decisão para os tomadores de decisão, os dados obtidos tanto das ferramentas de auditoria de código fonte quanto do método FAHP foram integrados na ferramenta Power BI(*Business Intelligence*) do fornecedor Microsoft®.

5.1 Modelo de Auditoria de Código Fonte para Tomada de Decisão

A Figura 5.1, apresenta os critérios de riscos por frequência de ocorrência em cada módulo do sistema, apresentando uma visão geral.

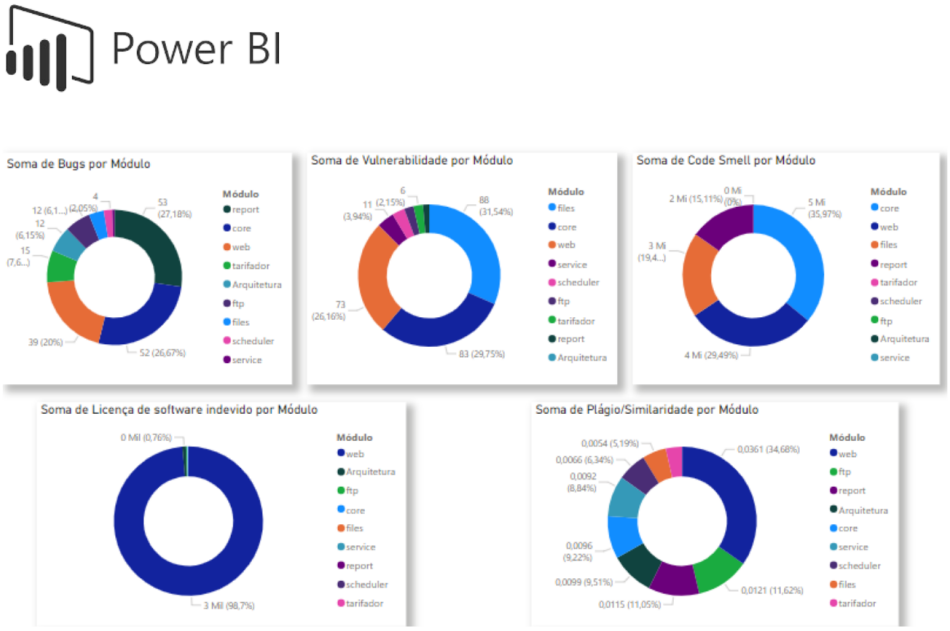


Figura 5.1: Painel de Critérios de Risco por Frequência de Ocorrência.
Fonte: Elaboração própria.

Na Figura 5.1, é possível observar a complexidade enfrentada pela equipe de desenvolvimento ao escolher quais riscos e módulos do software devem ter prioridade. Nota-se que, para cada critério de risco, a ordem de priorização varia entre os diferentes modelos do sistema, tornando desafiador para a equipe determinar por onde começar. Isso fortalece a necessidade de um modelo de apoio à decisão.

A Figura 5.2, apresenta o critério de risco Vulnerabilidade por frequência de ocorrência em cada módulo do sistema. Com isso, apresentando um visão deste risco.

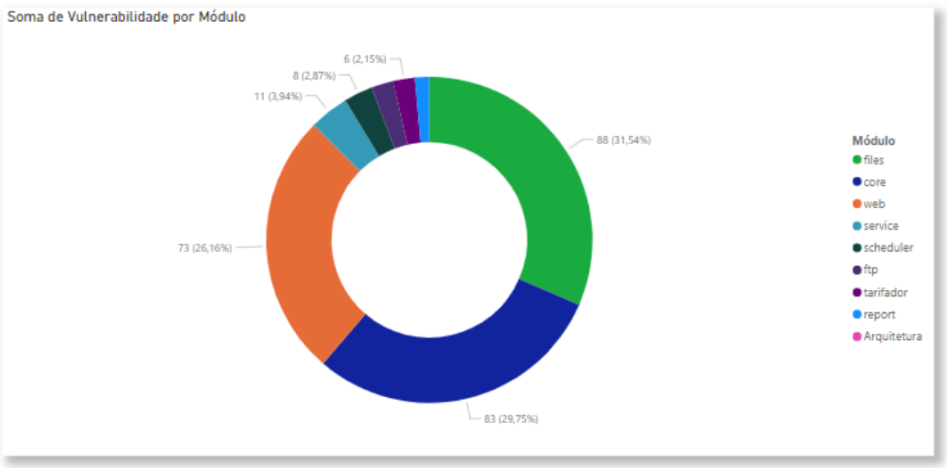


Figura 5.2: Painel de Vulnerabilidade por Frequência de Ocorrência.
Fonte: Elaboração própria.

No que diz respeito à vulnerabilidade, a análise da Figura 5.2 revela que o módulo File, seguido pelos módulos Core e Web, são os que apresentaram maior incidência de vulnerabilidades e requerem uma análise mais aprofundada.

A Figura 5.3, apresenta o critério de risco *Bugs* por frequência de ocorrência em cada módulo do sistema. Com isso, apresentando um visão deste risco.

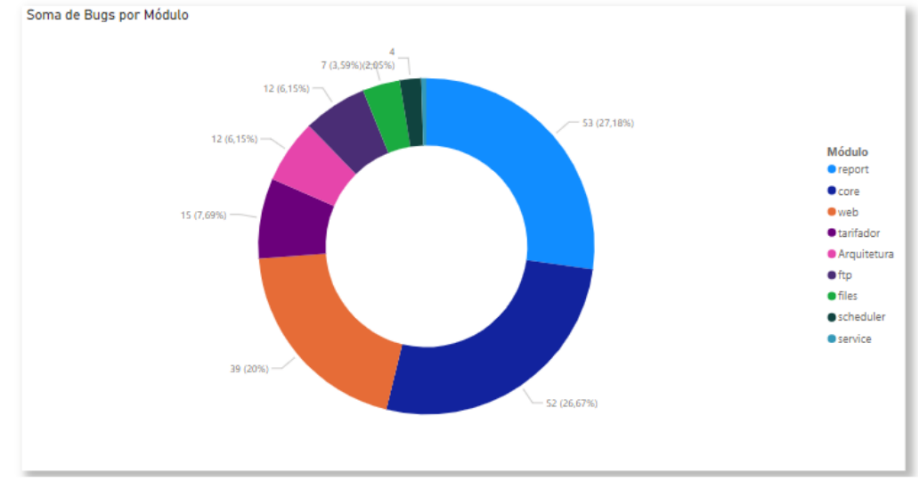


Figura 5.3: Painel de *Bugs* por Frequência de Ocorrência.
Fonte: Elaboração própria.

Já na análise de *Bugs*, podemos observar que na Figura 5.3 revela que o módulo Report, seguido pelos módulos Core e Web , são os que apresentaram maior incidência de *Bugs* e requerem uma análise mais aprofundada.

A Figura 5.4, apresenta o critério de risco *Code Smell* por frequência de ocorrência em cada módulo do sistema. Com isso, apresentando um visão deste risco.

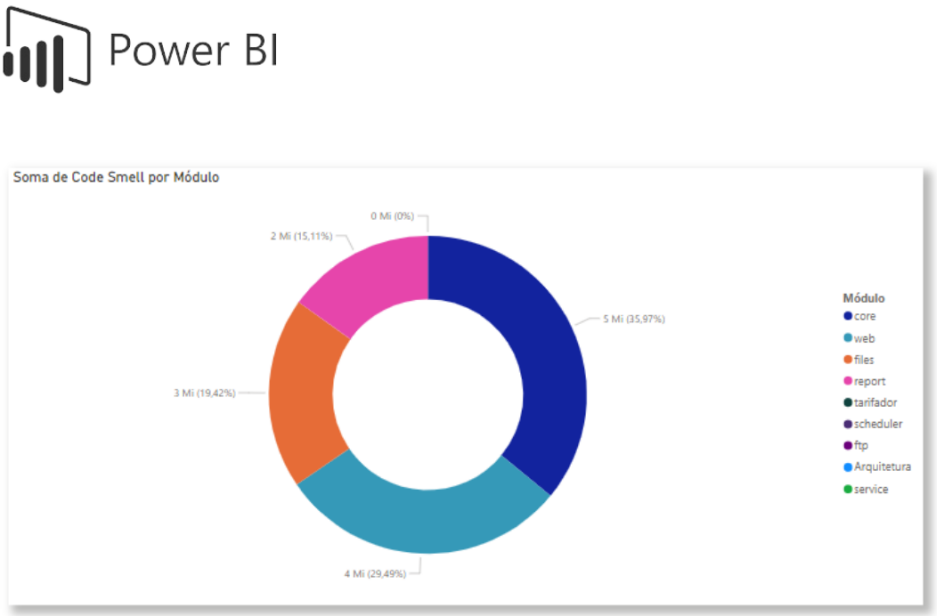


Figura 5.4: Painel de *Code Smell* por Frequência de Ocorrência.
Fonte: Elaboração própria.

No que diz respeito à *Code Smell*, a análise da Figura 5.4 revela que o módulo Core, seguido pelos módulos Web e Files, são os que apresentaram maior incidência de *Code Smell* e requerem uma análise mais aprofundada.

A Figura 5.5, apresenta o critério de risco de uso indevido de licença por frequência de ocorrência em cada módulo do sistema. Com isso, apresentando um visão deste risco.

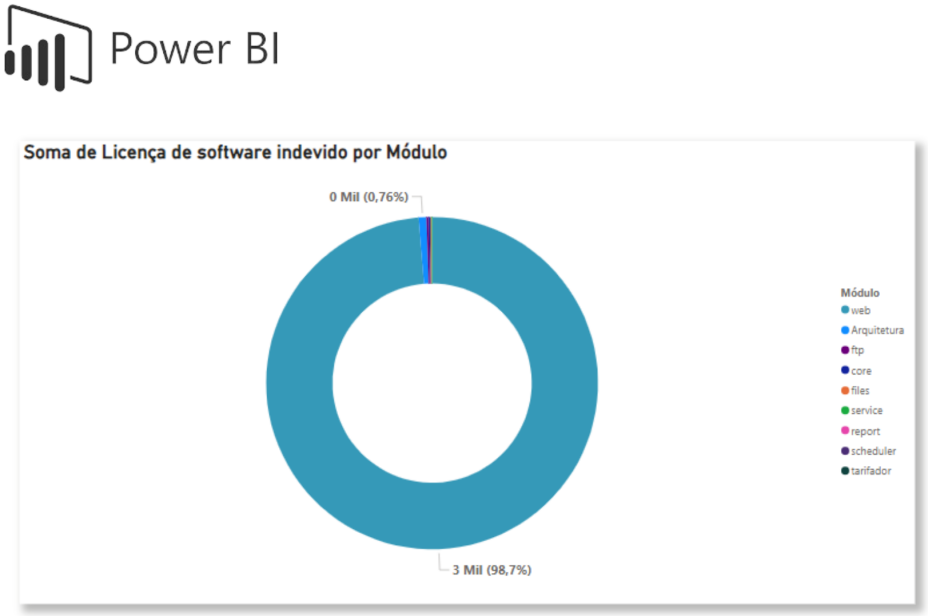


Figura 5.5: Painel de Uso Indevido por Licença por Frequência de Ocorrência.
Fonte: Elaboração própria.

Já na análise de Uso Indevido por Licença, podemos observar que na Figura 5.5 revela que o módulo Web, seguido pelos módulos Arquitetura e FTP , são os que apresentaram maior incidência de Uso Indevido por Licença e requerem uma análise mais aprofundada.

A Figura 5.6, apresenta o critério de risco de Plágio ou Similaridade por frequência de ocorrência em cada módulo do sistema. Com isso, apresentando um visão deste risco.

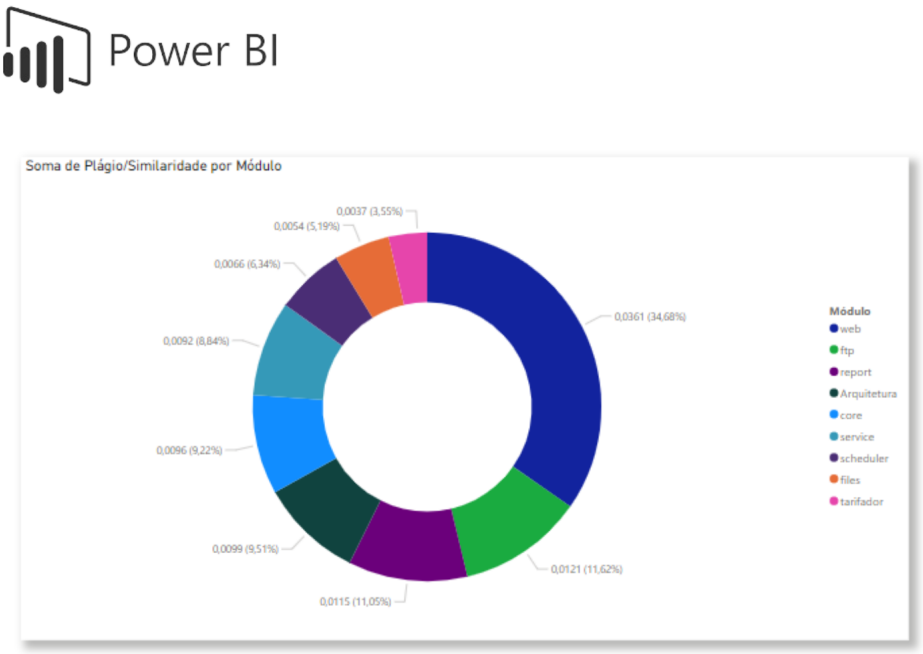


Figura 5.6: Painel de Plágio/Similaridade por Frequência de Ocorrência.
Fonte: Elaboração própria.

No que diz respeito à Plágio ou Similaridade, a análise da Figura 5.6 revela que o módulo Web, seguido pelos módulos FTP e Report, são os que apresentaram maior incidência de Plágio ou Similaridade e requerem uma análise mais aprofundada.

A Figura 5.7, apresenta o Ranking de Critérios de Risco e o Ranking de Priorização que necessitam de atenção.

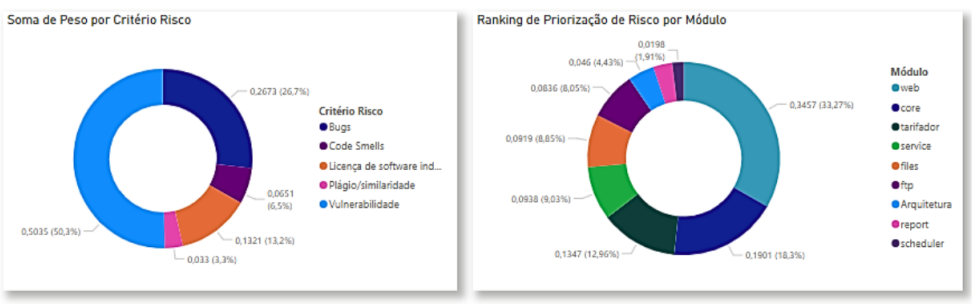


Figura 5.7: Painel de Ranking de Critérios de Risco e Ranking de Priorização.
Fonte: Elaboração própria.

Nesta Figura 5.7, é possível perceber a eficácia do modelo desenvolvido em orientar as decisões da equipe de desenvolvimento sobre quais riscos e módulos do software exigem atenção prioritária. O modelo organizou os critérios de risco, destacando que a vulnerabilidade requer tratamento imediato, com foco inicial no módulo Web.

A Figura 5.8, apresenta o Ranking de Critérios de Risco a serem tratados, ressaltando a necessidade de priorizar o risco de vulnerabilidade.

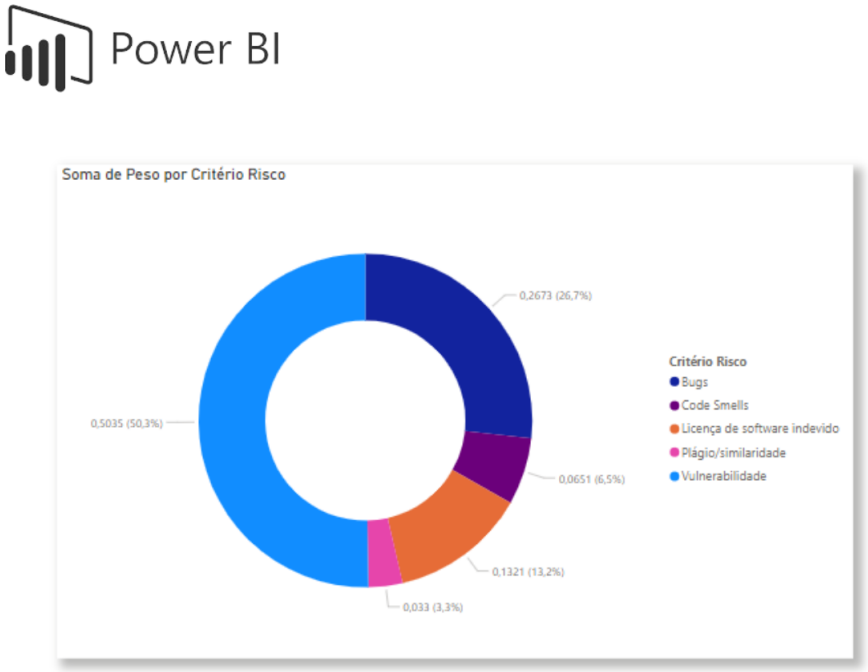


Figura 5.8: Painel de Ranking de Critérios de Risco.
Fonte: Elaboração própria.

No que diz respeito ao critérios de risco, a análise da Figura 5.8 revela que o risco Vulnerabilidade, seguido pelos Riscos *Bug* e Licença de software, são os que apresentaram maior risco e requerem uma análise mais aprofundada.

A Figura 5.9, apresenta o Racking de Priorização dos Maiores Riscos por Módulo do Software a serem tratados, ressaltando a necessidade de priorizar o módulo Web.

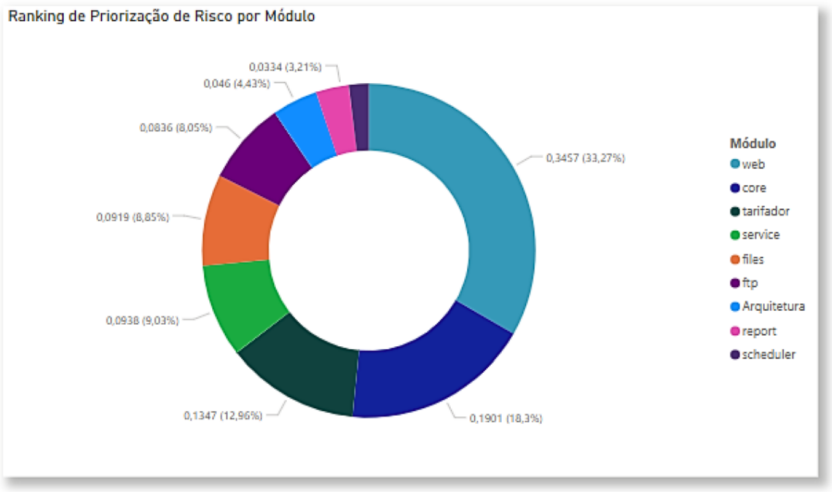


Figura 5.9: Painel de Ranking de Priorização do Maior Risco.
Fonte: Elaboração própria.

Já na análise dos módulos, podemos observar que a Figura 5.9 revela que o módulo Web, seguido pelos módulos Core e Tarifador , são os que apresentaram maior risco e requerem uma análise mais aprofundada.

5.2 Consideração Parcial

A Figura 5.7, apresenta o Ranking de Critérios de Risco com a Priorização dos Maiores Riscos por Módulo. É notável que a Vulnerabilidade é o critério com o maior risco. Ao observar a Figura 5.2, torna-se evidente que o módulo Files registra a maior quantidade de vulnerabilidades. No entanto, com o auxílio do modelo desenvolvido, foi identificado que o módulo que demanda priorização é o módulo Web, devido à sua maior probabilidade de impacto de risco nos negócios.

Capítulo 6

Conclusões Finais

Esta dissertação introduziu um modelo de apoio à decisão focado na priorização de riscos no processo de auditoria de código fonte. Este modelo foi testado por meio de um estudo de caso na empresa Ativu Tecnologia. Para alcançar o objetivo geral estabelecido, foram conduzidas atividades de identificação, seleção e definição dos critérios de risco relevantes para o contexto da auditoria de código fonte. Isso envolveu a realização de uma Revisão Sistemática da Literatura (RSL), que também contribuiu para a identificação e definição dos métodos e ferramentas de auditoria, bem como dos métodos multicritério apropriados, nomeadamente FAHP. Assim, os dois primeiros objetivos específicos desta pesquisa foram cumpridos.

Após a consecução desses objetivos específicos, a Construção e a Validação do modelo proposto foram finalizadas. Isso envolveu a aplicação dos métodos FAHP, bem como das abordagens do Índice de Consistência Relativa (ICR) e análise de sensibilidade, incorporando os *insights* e comentários fornecidos por especialistas, gerentes e analistas em relação à abordagem proposta nesta pesquisa.

Vale ressaltar que o comitê estratégico da empresa, com base neste trabalho, adotou o modelo desenvolvido e iniciou um projeto para tratativa dos riscos identificados. Conclui-se, portanto, que a simples posse de uma ferramenta automatizada de auditoria de código fonte para tratar os riscos não é suficiente. É crucial o planejamento dos dados e a visualização por meio de um modelo como o proposto aqui, que facilite a tomada de decisões e oriente em direção às soluções ideais.

6.1 Recomendações e Contribuições

O processo de auditoria de código fonte pode se tornar complexo e até oneroso quando não há uma estratégia definida para sua execução. Cada vez mais, os desenvolvedores estão

optando por avaliações superficiais em detrimento de uma análise mais aprofundada. No entanto, esta abordagem pode acarretar sérios problemas financeiros para as organizações.

Uma descoberta relevante, conforme indicada no estudo de caso de Damm et al. [139] no departamento de desenvolvimento, é que o custo de correção de falhas aumenta quase vinte vezes quando uma falha é detectada durante os testes do sistema em vez do teste inicial. Isso realça a necessidade de abordar os problemas na fase de programação, antes que o código seja implantado na produção.

O Modelo de Apoio à Decisão desenvolvido neste estudo se destaca por sua simplicidade e custo de implementação acessível, permitindo a identificação, análise, validação e tratamento eficientes dos critérios de risco, com o suporte de ferramentas de auditoria de código fonte. Adicionalmente, faz uso do método FAHP para facilitar a escolha das melhores alternativas de tratamento, especialmente em cenários complexos nos quais as opiniões dos especialistas podem divergir. No contexto do estudo de caso, as ferramentas selecionadas respeitaram as diretrizes e investimentos da empresa, mas é aconselhável que outras organizações adotem ferramentas automatizadas de acordo com suas normas, políticas e diretrizes. A essência é que o projeto esteja alinhado com as fases inspiradas no sistema de gerenciamento da ABNT NBR ISO 31000:2018 [140]. A Figura 6.1, adaptada dessa norma e alinhada com as fases do processo de gerenciamento de riscos, proporciona uma representação clara do modelo de auditoria de código fonte proposto neste estudo.

Figura 6.1: Sistema de Gerenciamento de Risco.
Fonte: Adaptada da ABNT NBR ISO 31000:2018 [140].

6.2 Delimitação

O foco do trabalho foi concentrado em software de aplicação Web. Os softwares de sistemas operacionais, firmware, aplicativos mobile ou outros tipos específicos ficaram fora do estudo.

Neste estudo, foi considerada a linguagem de programação Java, que é a principal utilizada no desenvolvimento do software do estudo de caso. No entanto, outras linguagens de programação como C++, Python, JavaScript entre outras não foram consideradas.

Neste trabalho, também não foi abordado o aspecto temporal, os custos e a matriz de riscos associados à correção dos problemas identificados. Essas informações em sua grande parte são tratadas no plano de execução do projeto e são consideradas confidenciais, fazendo parte do plano estratégico das organizações.

6.3 Sugestões para Trabalhos Futuros

Em estudos posteriores, outros modelos, como ANP ou ELECTRE ou TOPSIS, podem ser aplicados ao mesmo problema, e os resultados podem ser comparados. Uma outra sugestão é empregar o BPM para realizar uma análise de consistência e então comparar seus resultados com os obtidos neste estudo. Além disso, é viável criar modelos específicos para tratar cada classificação de risco, levando em consideração o grau de urgência e a estratégia da empresa. Em resumo, existem diversas opções disponíveis para priorizar riscos de software, a escolha deve ser feita com base na estratégia da organização.

Referências

[1] 360iResearch: *Market report.* Disponível em: `https://bit.ly/3TiPR29`. Acesso em: 17 de março 2024, 2024. 1, 32

[2] GVR: *Market analysis report.* Disponível em: `https://www.grandviewresearch.com/industry-analysis/software-market-report`. Acesso em: 05 de julho 2023, 2023. 1, 32

[3] Associação Brasileira de Empresas de Software: *Mercado brasileiro de software.* Disponível em: `https://bit.ly/4cdtzHT`. Acesso em: 05 de julho 2023, 2021. 1, 32

[4] Lucian Constantin: *What is devsecops? why it's hard to do well.* Disponível em: `https://www.csoonline.com/article/564095/`. Acesso em: 05 de julho 2023, 2020. 1, 32

[5] Petticrew, Mark e Helen Roberts: *Systematic reviews in the social sciences: A practical guide.* John Wiley & Sons, 2008. 3

[6] Rehman, Amjad e Tanzila Saba: *Evaluation of artificial intelligent techniques to secure information in enterprises.* Artificial Intelligence Review, 42:1029–1044, 2014. 4

[7] Sodiya, Adesina S, S Adebukola Onashoga e OB Ajayĩ: *Towards building secure software systems.* Issues in Informing Science & Information Technology, 3, 2006. 4

[8] Gilliam, David P, Thomas L Wolfe, Joseph S Sherif e Matt Bishop: *Software security checklist for the software life cycle.* Em *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, páginas 243–248. IEEE, 2003. 5

[9] Neamah, Karrar, Dzulkifli Mohamad, Tanzila Saba e Amjad Rehman: *Discriminative features mining for offline handwritten signature verification.* 3D Research, 5:1–6, 2014. 5

[10] Nunes, Francisco José Barreto, Arnaldo Dias Belchior e Adriano Bessa Albuquerque: *Security engineering approach to support software security.* Em *2010 6th World Congress on Services*, páginas 48–55. IEEE, 2010. 5

[11] Mundher, Myasar, Dzulkifli Muhamad, Amjad Rehman, Tanzila Saba e Firdous Kausar: *Digital watermarking for images security using discrete slantlet transform.* Applied Mathematics & Information Sciences, 8(6):2823, 2014. 5

[12] Jones, Russell L e Abhinav Rastogi: *Secure coding: building security into the software development life cycle.* Inf. Secur. J. A Glob. Perspect., 13(5):29–39, 2004. 5

[13] Citybank: *Violação do citi: 360 mil contas de cartão afetadas.* Disponível em: `https://g1.globo.com/tecnologia/noticia/2011/06/ataque-hacker-ao-citibank-afetou-360-mil-contas-de-clientes-do-banco.html`. Acesso em: 05 de julho 2023, 2011. 5, 32

[14] Emin Gun Sirer: *Thoughts on the dao hack dao ethereum.* Disponível em: `https://abes.com.br/wp-content/uploads/2021/08/ABES-EstudoMercadoBrasileirodeSoftware2021v02.pdf`. Acesso em: 05 de julho 2023, 2017. 5, 32

[15] Jackie Wattles: *Ataque de ransomware.* Disponível em: `https://money.cnn.com/2017/05/13/technology/ransomware-attack-who-got-hurt/index.html`. Acesso em: 05 de julho 2023, 2017. 5, 32

[16] IBM security: *Custos de violação de dados.* Disponível em: `https://ibm.co/3PrBBCZ`. Acesso em: 05 de julho 2023, 2019. 5, 32

[17] Mertzel, Nancy J: *Copying 0.03 percent of software code base not 'de minimis'.* Journal of Intellectual Property Law & Practice, 3(9):547–548, 2008. 5

[18] Mariscal, Gonzalo, Oscar Marban e Covadonga Fernandez: *A survey of data mining and knowledge discovery process models and methodologies.* The Knowledge Engineering Review, 25(2):137–166, 2010. 6

[19] Marconi, M de A e Eva Maria Lakatos: *Fundamentos de metodologia cientifica. 7 edição.* Rio de Janeiro: Editora Atlas, 2010. 6

[20] Prodanov, Cleber Cristiano e Ernani Cesar De Freitas: *Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição.* Editora Feevale, 2013. 6, 7

[21] Cervo, AL e PA Bervian: *Metodologia científica. 5aed*, 2002. 6

[22] Triperina, Evangelia: *Visual interactive knowledge management for multicriteria decision making and ranking in linked open data environments.* Tese de Doutoramento, Limoges, 2020. 6

[23] Gerhardt, Tatiana Engel e Denise Tolfo Silveira: *Métodos de Pesquisa.* Plageder, 2009. 7

[24] Dam, Hoa Khanh, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy e Aditya Ghose: *Automatic feature learning for vulnerability prediction.* arXiv preprint arXiv:1708.02368, 2017. 11

[25] Joy, Mike e Michael Luck: *Plagiarism in programming assignments.* IEEE Transactions on Education, 42(2):129–133, 1999. 11, 15

[26] Butakov, Sergey, Marina Kim e Svetlana Kim: *Low ram footprint algorithm for small scale plagiarism detection projects.* Em *2012 International Conference on Information Science and Applications*, páginas 1–2. IEEE, 2012. 11, 16

[27] Positive Technologies: *Ameaças e vulnerabilidades em aplicativos da web 2020–2021.* Disponível em: `https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021/`. Acesso em: 05 de julho 2023, 2022. 12, 13, 32

[28] OWASP: *Os 10 principais riscos de segurança de aplicativos da web.* Disponível em: `https://owasp.org/www-project-top-ten/`. Acesso em: 05 de julho 2023. 12, 13, 32

[29] Fowler, Martin, Kent Beck, John Brant, William Opdyke e Don Refactoring Roberts: *Improving the design of existing code.* Addisonâ‾'Wesley, 6, 1999. 13, 14, 15

[30] Brown, William H, Raphael C Malveau, Hays W" Skip" McCormick e Thomas J Mowbray: *AntiPatterns: refactoring software, architectures, and projects in crisis.* John Wiley & Sons, Inc., 1998. 14

[31] Ordonez, Carlos, Zhibo Chen e Javier García-García: *Metadata management for federated databases.* Em *Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience*, páginas 31–38, 2007. 15

[32] Gerlec, Črt e Marjan Heričko: *The impact of structural source code changes on software quality.* Em *AIP Conference Proceedings*, páginas 470–473. American Institute of Physics, 2012. 15

[33] Chuda, Daniela, Pavol Navrat, Bianka Kovacova e Pavel Humay: *The issue of (software) plagiarism: A student view.* IEEE Transactions on Education, 55(1):22–28, 2011. 16

[34] German, Daniel M e Ahmed E Hassan: *License integration patterns: Addressing license mismatches in component-based development.* Em *2009 IEEE 31st international conference on software engineering*, páginas 188–198. IEEE, 2009. 16

[35] Courts, United States. Congress. House. Committee on the Judiciary. Subcommittee on e Intellectual Property: *United States Copyright Office and Sound Recordings as Work Made for Hire: Hearing Before the Subcommittee on Courts and Intellectual Property of the Committee on the Judiciary, House of Representatives, One Hundred Sixth Congress, Second Session, May 25, 2000.* US Government Printing Office, 2000. 16

[36] Rosen, Lawrence: *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice Hall PTR, New Jersey, USA, 2004. 17

[37] Rodau, Andrew G: *Protecting computer software: After apple computer, inc. v. franklin computer corp., 714 f. 2d 1240 (3d cir. 1983), does copyright provide the best protection.* Temp. LQ, 57:527, 1984. 17

[38] Goldstein, Paul: *International copyright: principles, law, and practice*. Oxford University Press, USA, 2001. 17

[39] Lai, Stanley: *Copyright Protection of Computer Software in the United Kingdom*. Bloomsbury Publishing, 2000. 17

[40] German, Daniel M, Massimiliano Di Penta, Yann Gael Gueheneuc e Giuliano Antoniol: *Code siblings: Technical and legal implications of copying code between applications*. Em *2009 6th IEEE International Working Conference on Mining Software Repositories*, páginas 81–90. IEEE, 2009. 17, 21

[41] Yamaguchi, Fabian, Konrad Rieck *et al.*: *Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning*. Em *5th USENIX Workshop on Offensive Technologies (WOOT 11)*, 2011. 18

[42] Lingzi, Xiang e Lin Zhi: *An overview of source code audit*. Em *2015 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration*, páginas 26–29. IEEE, 2015. 18, 19

[43] Eaddy, Marc, Alfred V Aho, Giuliano Antoniol e Yann Gaël Guéhéneuc: *Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis*. Em *2008 16th IEEE International Conference on Program Comprehension*, páginas 53–62. IEEE, 2008. 18

[44] Xiaomeng, Wang, Zhang Tao, Xin Wei e Hou Changyu: *A survey on source code review using machine learning*. Em *2018 3rd International Conference on Information Systems Engineering (ICISE)*, páginas 56–60. IEEE, 2018. 18, 19

[45] Lacombe, Guilhem, David Feliot, Etienne Boespflug e Marie Laure Potet: *Combining static analysis and dynamic symbolic execution in a toolchain to detect fault injection vulnerabilities*. Journal of Cryptographic Engineering, páginas 1–18, 2023. 19

[46] Brooks, Teresa Nicole: *Survey of automated vulnerability detection and exploit generation techniques in cyber reasoning systems*. Em *Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2*, páginas 1083–1102. Springer, 2019. 19

[47] Xiaomeng, Wang, Zhang Tao, Wu Runpu, Xin Wei e Hou Changyu: *Cpgva: Code property graph based vulnerability analysis by deep learning*. Em *2018 10th International Conference on Advanced Infocomm Technology (ICAIT)*, páginas 184–188. IEEE, 2018. 19

[48] Avgeriou, Paris C, Davide Taibi, Apostolos Ampatzoglou, Francesca Arcelli Fontana, Terese Besker, Alexander Chatzigeorgiou, Valentina Lenarduzzi, Antonio Martini, Athanasia Moschou, Ilaria Pigazzini *et al.*: *An overview and comparison of technical debt measurement tools*. IEEE software, 38(3):61–71, 2020. 19

[49] Marcilio, Diego, Rodrigo Bonifácio, Eduardo Monteiro, Edna Canedo, Welder Luz e Gustavo Pinto: *Are static analysis violations really fixed? a closer look at realistic usage of sonarqube*. Em *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, páginas 209–219. IEEE, 2019. 19

[50] SonarQUBE: *Sonarqube 10.1 documentação*. Disponível em: `https://docs.sonarsource.com/sonar`. Acesso em: 05 de julho 2023, 2023. 20, 32, 41, 42, 43

[51] Gobeille, Robert: *The fossology project*. Em *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, páginas 47–50, 2008. 21

[52] Teixeira, Clever Marcos, Marcelo Augusto Cicogna e Maurício Rodrigues Morais: *Software para detecção de textos com plágio baseado em busca pela internet*. Em *Congresso Nacional de Iniciação Científica*, volume 11, 2011. 22

[53] CopySpider: *Copyspider*. Disponível em: `https://copyspider.com.br/main/pt-br/history`. Acesso em: 21 de setembro 2023, 2023. 23, 32, 48, 49, 50, 51, 52

[54] ABNT, ISO GUIA: *73: 2009-gestão de riscos–vocabulário*. Rio de Janeiro: Associação Brasileira de Normas Técnicas, 2009. 23

[55] Kumar, Abhishek, Bikash Sah, Arvind R Singh, Yan Deng, Xiangning He, Praveen Kumar e RC Bansal: *A review of multi criteria decision making (mcdm) towards sustainable renewable energy development*. Renewable and Sustainable Energy Reviews, 69:596–609, 2017. 23

[56] Belton, Valerie e Theodor Stewart: *Multiple criteria decision analysis: an integrated approach*. Springer Science & Business Media, 2002. 24, 63

[57] Wang, Jia Wen, Ching Hsue Cheng e Kun Cheng Huang: *Fuzzy hierarchical topsis for supplier selection*. Applied Soft Computing, 9(1):377–386, 2009. 24, 25

[58] Wind, Yoram e Thomas L Saaty: *Marketing applications of the analytic hierarchy process*. Management Science, 26(7):641–658, 1980. 24, 61

[59] Wernke, Rodney e Antonio Cezar Bornia: *A contabilidade gerencial e os métodos multicriteriais*. Revista Contabilidade & Finanças, 12:60–71, 2001. 25

[60] Önüt, Semih, Selin Soner Kara e Elif Işik: *Long term supplier selection using a combined fuzzy mcdm approach: A case study for a telecommunication company*. Expert Systems with Applications, 36(2):3887–3895, 2009. 25

[61] Hwang, C L e Abu Syed Md Masud: *Multiple objective decision making—methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media, 2012. 25

[62] Benayoun, Raphael, Bernard Roy e Barry Sussman: *Electre: Une méthode pour guider le choix en présence de points de vue multiples*. Note de travail, 49:2–120, 1966. 25

[63] Cavallaro, Fausto: *A comparative assessment of thin-film photovoltaic production processes using the electre iii method.* Energy Policy, 38(1):463–474, 2010. 25

[64] Li, Hui e Jie Sun: *Business failure prediction using hybrid2 case-based reasoning (h2cbr).* Computers & Operations Research, 37(1):137–151, 2010. 25

[65] Brans, Jean Pierre, R Nadeau e M Landry: *L'ingénierie de la décision.* Elaboration d'instruments d'aide à la décision. La méthode PROMETHEE. In l'Aide à la Décision: Nature, Instruments et Perspectives d'Avenir, páginas 183–213, 1982. 25

[66] Vincke, Jean Pierre e Ph Brans: *A preference ranking organization method. the promethee method for mcdm.* Management Science, 31(6):647–656, 1985. 25

[67] Brans, Jean Pierre, Ph Vincke e Bertrand Mareschal: *How to select and how to rank projects: The promethee method.* European journal of operational research, 24(2):228–238, 1986. 25

[68] Goguen, Joseph A: *La zadeh. fuzzy sets. information and control, vol. 8 (1965), pp. 338–353.-la zadeh. similarity relations and fuzzy orderings. information sciences, vol. 3 (1971), pp. 177–200.* The Journal of Symbolic Logic, 38(4):656–657, 1973. 25

[69] Cheng, Ching Hsue: *Evaluating naval tactical missile systems by fuzzy ahp based on the grade value of membership function.* European journal of operational research, 96(2):343–350, 1997. 25

[70] Cheng, Ching Hsue, Kuo Lung Yang e Chia Lung Hwang: *Evaluating attack helicopters by ahp based on linguistic variable weight.* European journal of operational research, 116(2):423–435, 1999. 25

[71] Ruoning, Xu e Zhai Xiaoyan: *Extensions of the analytic hierarchy process in fuzzy environment.* Fuzzy sets and Systems, 52(3):251–257, 1992. 25

[72] Petkovic, Jasna, Zoran Sevarac, Maja Levi Jaksic e Sanja Marinkovic: *Application of fuzzy ahp method for choosing a technology within service company.* Technics technologies education management-ttem, 7(1):332–341, 2012. 25

[73] Van Laarhoven, Peter JM e Witold Pedrycz: *A fuzzy extension of saaty's priority theory.* Fuzzy sets and Systems, 11(1-3):229–241, 1983. 25

[74] Buckley, James J: *Fuzzy hierarchical analysis.* Fuzzy sets and systems, 17(3):233–247, 1985. 25, 27, 56

[75] Chang, Da Yong: *Applications of the extent analysis method on fuzzy ahp.* European journal of operational research, 95(3):649–655, 1996. 25

[76] Kilincci, Ozcan e Suzan Aslı Onal: *Fuzzy ahp approach for supplier selection in a washing machine company.* Expert systems with Applications, 38(8):9656–9664, 2011. 25, 55

[77] Chou, Shih Wei e Yu Chieh Chang: *The implementation factors that influence the erp (enterprise resource planning) benefits.* Decision support systems, 46(1):149–157, 2008. 27

[78] Page, Matthew J, David Moher, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan *et al.*: *Prisma 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews*. bmj, 372, 2021. 29, 30

[79] Wang, Jingjing, Minhuan Huang, Yuanping Nie e Jin Li: *Static analysis of source code vulnerability using machine learning techniques: A survey*. Em *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, páginas 76–86. IEEE, 2021. 32

[80] Zhang, Gen, Peng Fei Wang, Tai Yue, Xiang Dong Kong, Xu Zhou e Kai Lu: *ovaflow: Detecting memory corruption bugs with fuzzing-based taint inference*. Journal of Computer Science and Technology, 37(2):405–422, 2022. 32

[81] AbuHassan, Amjad, Mohammad Alshayeb e Lahouari Ghouti: *Detection of design smells using adaptive neuro-fuzzy approaches*. International Journal of Fuzzy Systems, 24(4):1927–1943, 2022. 32

[82] Schreiber, Andreas, Tim Sonnekalb e Lynn von Kurnatowski: *Towards visual analytics dashboards for provenance-driven static application security testing*. Em *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, páginas 42–46. IEEE, 2021. 32, 33

[83] Filus, Katarzyna, Paweł Boryszko, Joanna Domańska, Miltiadis Siavvas e Erol Gelenbe: *Efficient feature selection for static analysis vulnerability prediction*. Sensors, 21(4):1133, 2021. 32, 33

[84] d'Aragona, Dario Amoroso, Fabiano Pecorelli, Maria Teresa Baldassarre, Davide Taibi e Valentina Lenarduzzi: *Technical debt diffuseness in the apache ecosystem: A differentiated replication*. Em *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, páginas 825–833. IEEE, 2023. 32, 33

[85] Do Xuan, Cho, Dao Hoang Mai, Ma Cong Thanh e Bui Van Cong: *A novel approach for software vulnerability detection based on intelligent cognitive computing*. The Journal of Supercomputing, 79(15):17042–17078, 2023. 32, 33

[86] Singh, Malya e Vishal Gupta: *Review of extrinsic plagiarism detection techniques and their efficiency comparison*. Em *International Conference on Advanced Network Technologies and Intelligent Computing*, páginas 609–624. Springer, 2021. 32, 34

[87] Pang, Ashley e Frank Vahid: *Variability-inducing requirements for programs: Increasing solution variability for similarity checking*. Em *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, páginas 430–435, 2023. 32, 34

[88] Woo, Seunghoon, Sunghan Park, Seulbae Kim, Heejo Lee e Hakjoo Oh: *Centris: A precise and scalable approach for identifying modified open-source software reuse*. Em *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, páginas 860–872. IEEE, 2021. 32, 34

[89] Liu, Xiao e Gyun Woo: *Using sonarqube to enhance the accuracy of programming assignment evaluation.* Em *INTED2019 Proceedings*, páginas 5262–5267. IATED, 2019. 32, 33

[90] Wang, Junjie, Yuchao Huang, Song Wang e Qing Wang: *Find bugs in static bug finders.* Em *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, páginas 516–527, 2022. 32, 33

[91] Saborido, Rubén, Javier Ferrer, Francisco Chicano e Enrique Alba: *Automatizing software cognitive complexity reduction.* IEEE Access, 10:11642–11656, 2022. 32, 33

[92] Hegedűs, Péter e Rudolf Ferenc: *Static code analysis alarms filtering reloaded: A new real-world dataset and its ml-based utilization.* IEEE Access, 10:55090–55101, 2022. 32, 33

[93] Liu, Zhijuan, Li Zhang, Xuangou Wu e Wei Zhao: *Test case filtering based on generative adversarial networks.* Em *2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR)*, páginas 65–69. IEEE, 2022. 32, 33

[94] Xu, Sihan, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu e Hua Ji: *Lidetector: License incompatibility detection for open source software.* ACM Transactions on Software Engineering and Methodology, 32(1):1–28, 2023. 32, 33

[95] Moraes, Joao Pedro, Ivanilton Polato, Igor Wiese, Filipe Saraiva e Gustavo Pinto: *From one to hundreds: multi-licensing in the javascript ecosystem.* Empirical Software Engineering, 26:1–29, 2021. 32, 34

[96] Higashi, Yunosuke, Yuki Manabe e Masao Ohira: *Clustering oss license statements toward automatic generation of license rules.* Em *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, páginas 30–35. IEEE, 2016. 32, 33

[97] Novak, Matija, Mike S Joy e Olfat M Mirza: *Improved plagiarism detection with collaboration network visualization based on source-code similarity.* Em *2021 IEEE Technology & Engineering Management Conference-Europe (TEMSCON-EUR)*, páginas 1–6. IEEE, 2021. 32, 34

[98] Hirsch, Thomas e Birgit Hofer: *Identifying non-natural language artifacts in bug reports.* Em *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, páginas 191–197. IEEE, 2021. 32

[99] Karnalim, Oscar, Simon, William Chivers e Billy Susanto Panca: *Educating students about programming plagiarism and collusion via formative feedback.* ACM Transactions on Computing Education (TOCE), 22(3):1–31, 2022. 32, 34

[100] Akremi, Aymen: *Software security static analysis false alerts handling approaches.* International Journal of Advanced Computer Science and Applications, 12(11):702–711, 2021. 32, 33

[101] Kapitsaki, Georgia M e Georgia Charalambous: *Find your open source license now!* Em *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, páginas 1–8. IEEE, 2016. 32, 33

[102] Kapitsaki, Georgia M e Georgia Charalambous: *Modeling and recommending open source licenses with findosslicense.* IEEE Transactions on Software Engineering, 47(5):919–935, 2019. 32, 33

[103] Gamalielsson, Jonas e Björn Lundell: *On licensing and other conditions for contributing to widely used open source projects: an exploratory analysis.* Em *Proceedings of the 13th International Symposium on Open Collaboration*, páginas 1–14, 2017. 32, 33

[104] Vendome, Christopher, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M German e Denys Poshyvanyk: *When and why developers adopt and change software licenses.* Em *2015 IEEE international conference on software maintenance and evolution (ICSME)*, páginas 31–40. IEEE, 2015. 32, 34

[105] Vendome, Christopher, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German e Denys Poshyvanyk: *Machine learning-based detection of open source license exceptions.* Em *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, páginas 118–129. IEEE, 2017. 32, 34

[106] Caneill, Matthieu, Daniel M Germán e Stefano Zacchiroli: *The debsources dataset: two decades of free and open source software.* Empirical Software Engineering, 22:1405–1437, 2017. 32, 34

[107] Serafini, Daniele e Stefano Zacchiroli: *Efficient prior publication identification for open source code.* Em *Proceedings of the 18th International Symposium on Open Collaboration*, páginas 1–8, 2022. 32, 34

[108] Qiu, Shi, Daniel M German e Katsuro Inoue: *Empirical study on dependency-related license violation in the javascript package ecosystem.* Journal of Information Processing, 29:296–304, 2021. 32, 34

[109] Karnalim, Oscar, Simon e William Chivers: *Layered similarity detection for programming plagiarism and collusion on weekly assessments.* Computer Applications in Engineering Education, 30(6):1739–1752, 2022. 32, 34

[110] Karnalim, Oscar *et al.*: *Explanation in code similarity investigation.* IEEE Access, 9:59935–59948, 2021. 32, 34

[111] Hrkút, Patrik, Michal Ďuračík, Štefan Toth e Matej Meško: *Current trends in the search for similarities in source codes with an application in the field of plagiarism and clone detection.* Em *2023 33rd Conference of Open Innovations Association (FRUCT)*, páginas 77–84. IEEE, 2023. 32, 34

[112] Anjali, V, TR Swapna e Bharat Jayaraman: *Plagiarism detection for java programs without source codes.* Procedia Computer Science, 46:749–758, 2015. 32, 34

[113] Nazim, Mohammad Taneem Bin, Md Jobair Hossain Faruk, Hossain Shahriar, Md Abdullah Khan, Mohammad Masum, Nazmus Sakib e Fan Wu: *Systematic analysis of deep learning model for vulnerable code detection.* Em *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, páginas 1768–1773. IEEE, 2022. 32, 33

[114] Qiu, Shi, German M Daniel e Katsuro Inoue: *A machine learning method for automatic copyright notice identification of source files.* IEICE TRANSACTIONS on Information and Systems, 103(12):2709–2712, 2020. 32, 33

[115] Jin, Xin, Hongyan Li, Hengwang Liu, Wen Wang e Xin Sun: *Intelligent vulnerability association algorithm based on association rule mining.* Em *International Conference on Big Data Analytics for Cyber-Physical System in Smart City*, páginas 455–462. Springer, 2022. 32, 33

[116] Jain, Pooja, Vaibhav Agasti e Tapan Kumar: *Plagiarism detection of online submissions using high level fuzzy petri nets in pandemic times.* Em *Communication, Networks and Computing: Second International Conference, CNC 2020, Gwalior, India, December 29–31, 2020, Revised Selected Papers 2*, páginas 72–86. Springer, 2021. 32, 34

[117] Alqarni, Mansour, Akramul Azim e Tegveer Singh: *Evdd-a novel dataset for embedded system vulnerability detection mechanism.* Em *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, páginas 764–769. IEEE, 2022. 32

[118] Papoutsoglou, Maria, Georgia M Kapitsaki, Daniel German e Lefteris Angelis: *An analysis of open source software licensing questions in stack exchange sites.* Journal of Systems and Software, 183:111113, 2022. 32, 33

[119] Smith, W Spencer, D Adam Lazzarato e Jacques Carette: *State of the practice for mesh generation and mesh processing software.* Advances in Engineering Software, 100:53–71, 2016. 32, 35

[120] Yang, Yunxue, Shuyuan Jin e Xiaowei He: *Software vulnerability severity evaluation based on economic losses.* Em *Trustworthy Computing and Services: International Conference, ISCTCS 2014, Beijing, China, November 28-29, 2014, Revised Selected papers*, páginas 144–151. Springer, 2015. 32, 35

[121] Cui, Can, Bin Liu, Peng Xiao e Shihai Wang: *Can defect prediction be useful for coarse-level tasks of software testing?* Applied Sciences, 10(15):5372, 2020. 32, 35

[122] Anjum, Misbah, Vernika Agarwal, PK Kapur e Sunil Kumar Khatri: *Two-phase methodology for prioritization and utility assessment of software vulnerabilities.* International Journal of System Assurance Engineering and Management, 11:289–300, 2020. 32, 35

[123] Song, Jialiang, Jihong Han, Danlin Zhang, Lin Yuan e Lulu Shao: *Evaluation of security vulnerability severity based on cmahp.* Em *2016 2nd IEEE International*

Conference on Computer and Communications (ICCC), páginas 1056–1060. IEEE, 2016. 32, 35

[124] Abushark, Yoosef B, A Irshad Khan, Fawaz Alsolami, Abdulmohsen Almalawi, Md Mottahir Alam, Alka Agrawal, Rajeev Kumar e Raees Ahmad Khan: *Cyber security analysis and evaluation for intrusion detection systems.* Comput. Mater. Contin, 72:1765–1783, 2022. 32, 35

[125] Agrawal, Alka, Adil Hussain Seh, Abdullah Baz, Hosam Alhakami, Wajdi Alhakami, Mohammed Baz, Rajeev Kumar e Raees Ahmad Khan: *Software security estimation using the hybrid fuzzy anp-topsis approach: Design tactics perspective.* Symmetry, 12(4):598, 2020. 32, 35

[126] Seh, Adil Hussain, Jehad F Al-Amri, Ahmad F Subahi, Md Tarique Jamal Ansari, Rajeev Kumar, Mohammad Ubaidullah Bokhari e Raees Ahmad Khan: *Hybrid computational modeling for web application security assessment.* CMC-Comput., Mater. Continua, 70(1):469–489, 2022. 32, 35

[127] Bhatt, Navneet, Jasmine Kaur, Adarsh Anand e Omar H Alhazmi: *Selecting best software vulnerability scanner using intuitionistic fuzzy set topsis.* Computers, Materials & Continua, 72(2), 2022. 32, 35

[128] Goyal, Anjali: *Effective bug triage for non-reproducible bugs.* Em *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, páginas 487–488. IEEE, 2017. 32, 35

[129] BRASIL LEI Nº 13.709, DE 14 DE AGOSTO DE 2018: *Lei geral de proteção de dados pessoais (lgpd).* Disponível em: `http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm`. Acesso em: 05 de julho 2023, 2018. 32

[130] Fossology: *Fossology.* Disponível em: `https://www.fossology.org`. Acesso em: 21 de setembro 2023, 2023. 32, 44, 45, 46, 47

[131] rayyan: *rayyan.* Disponível em: `https://www.rayyan.ai/`. Acesso em: 21 de setembro 2023, 2023. 32

[132] Prof. Valdecy Pereira, D.Sc: *pydecisions - fuzzy ahp (fuzzy analytic hierarchy process).* Disponível em: `https://colab.research.google.com/drive/1RtEMOLGL5wtmheMRZv8emcO5wbjYVBCo?usp=sharing`. Acesso em: 21 de setembro 2023, 2023. 32, 57, 63, 90, 91, 92, 93, 94, 95, 96

[133] Marttunen, Mika, Fridolin Haag, Valerie Belton, Jyri Mustajoki e Judit Lienert: *Methods to inform the development of concise objectives hierarchies in multi-criteria decision analysis.* European Journal of Operational Research, 277(2):604–620, 2019. 53

[134] Saaty, Thomas L: *A scaling method for priorities in hierarchical structures.* Journal of mathematical psychology, 15(3):234–281, 1977. 54

[135] Grenning, James: *Planning poker or how to avoid analysis paralysis while release planning*. Hawthorn Woods: Renaissance Software Consulting, 3:22–23, 2002. 57, 63

[136] Gunaydin, H Murat: *The delphi method*. Optimization Group, 2006. 57

[137] Qureshi, ME, Steve R Harrison e MK Wegener: *Validation of multicriteria analysis models*. Agricultural Systems, 62(2):105–116, 1999. 61, 63

[138] Triantaphyllou, Evangelos e Alfonso Sánchez: *A sensitivity analysis approach for some deterministic multi-criteria decision-making methods*. Decision sciences, 28(1):151–194, 1997. 63

[139] Damm, LO, L Lundberg e C Wohlin: *Phase-oriented process assessment using fault analysis*. Em *Proceedings of the 11th European Conference on Software Process Improvement, Springer-Verlag*, 2004. 76

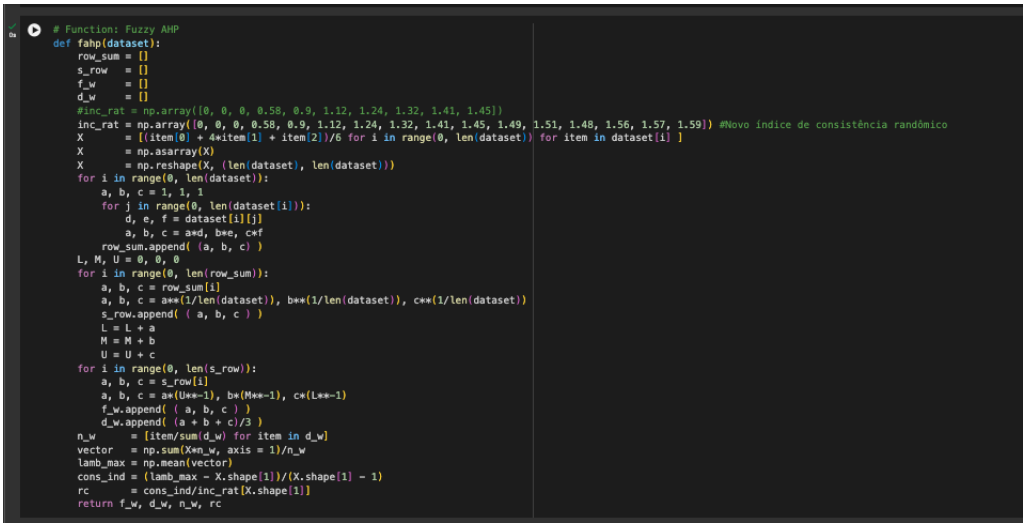
[140] ISO, IEC: *Abnt nbr iso/iec 31000: 2018 gestão de riscos-diretrizes*. Associação Brasileira de Normas Técnicas, 2018. 76

Apêndice A

Código Fonte do Modelo FAHP Desenvolvido em Python

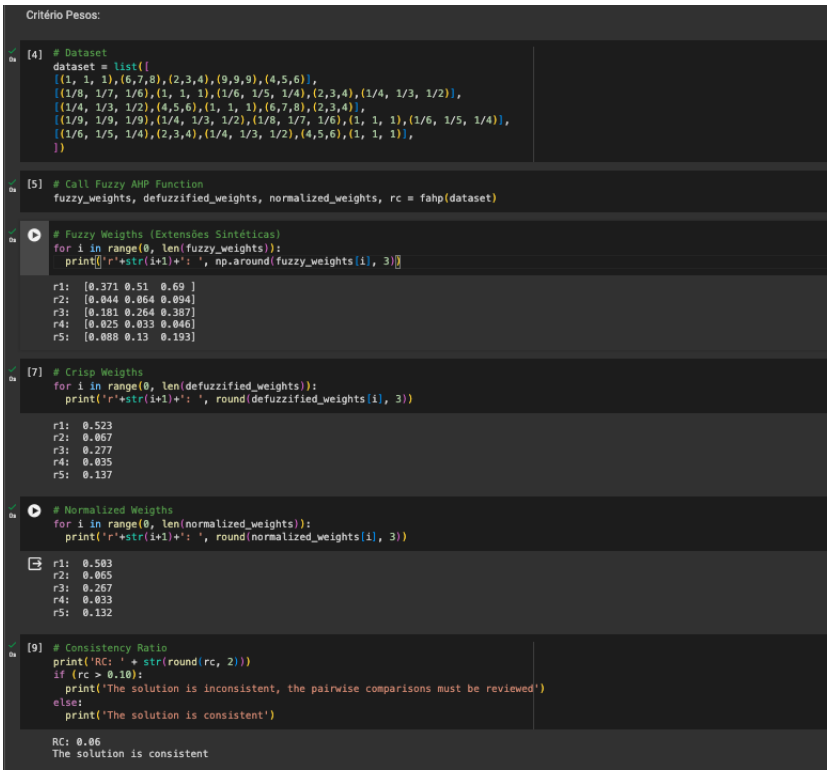
```
!pip install pyDecision
!pip install numpy
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyDecision) (2.8.2)
Requirement already satisfied: anyio<4,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai->pyDecision) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai->pyDecision) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai->pyDecision)
  Downloading httpx-0.25.2-py3-none-any.whl (74 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.0/75.0 kB 11.2 MB/s eta 0:00:00
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai->pyDecision) (1.3.0)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai->pyDecision) (4.66.1)
Requirement already satisfied: typing-extensions<5,>=4.5 in /usr/local/lib/python3.10/dist-packages (from openai->pyDecision) (4.5.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pyDecision) (2023.3.post1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pyDecision) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pyDecision) (3.2.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.5.0->openai->pyDecision) (3.6)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.5.0->openai->pyDecision) (1.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->openai->pyDecision) (2023.11.17)
Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai->pyDecision)
  Downloading httpcore-1.0.2-py3-none-any.whl (76 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.9/76.9 kB 12.1 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai->pyDecision)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 9.5 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->pyDecision) (1.16.0)
Requirement already satisfied: aiohttp<4.0,>=3.0 in /usr/local/lib/python3.10/dist-packages (from cohere->llmx->pyDecision) (3.9.1)
Collecting backoff<3.0,>=2.0 (from cohere->llmx->pyDecision)
  Downloading backoff-2.2.1-py3-none-any.whl (15 kB)
Collecting fastavro<2.0,>=1.8 (from cohere->llmx->pyDecision)
  Downloading fastavro-1.9.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 39.7 MB/s eta 0:00:00
Collecting importlib_metadata<7.0,>=6.0 (from cohere->llmx->pyDecision)
  Downloading importlib_metadata-6.11.0-py3-none-any.whl (23 kB)
Requirement already satisfied: requests<3.0.0,>=2.25.0 in /usr/local/lib/python3.10/dist-packages (from cohere->llmx->pyDecision) (2.31.0)
Requirement already satisfied: urllib3<3,>=1.26 in /usr/local/lib/python3.10/dist-packages (from cohere->llmx->pyDecision) (2.0.7)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google.auth->llmx->pyDecision) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google.auth->llmx->pyDecision) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google.auth->llmx->pyDecision) (4.9)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.10/dist-packages (from tiktoken->llmx->pyDecision) (2023.6.3)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer->llmx->pyDecision) (8.1.7)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (1.9.3)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere->llmx->pyDecision) (4.0.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib_metadata<7.0,>=6.0->cohere->llmx->pyDecision) (3.17.0)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google.auth->llmx->pyDecision) (0.5.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.25.0->cohere->llmx->pyDecision) (3.3.2)
Installing collected packages: importlib_metadata, h11, fastavro, backoff, tiktoken, httpcore, httpx, cohere, openai, pyDecision
  Attempting uninstall: importlib_metadata
    Found existing installation: importlib-metadata 7.0.0
    Uninstalling importlib-metadata-7.0.0:
      Successfully uninstalled importlib-metadata-7.0.0
Successfully installed backoff-2.2.1 cohere-4.37 fastavro-1.9.1 h11-0.14.0 httpcore-1.0.2 httpx-0.25.2 importlib_metadata-6.11.0 openai-1.3.7 pyDecision-4.3.9 tiktoken-0.5.2
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)

[2] # Required Libraries
    import numpy as np

    # Fuzzy AHP
    from pyDecision.algorithm import fuzzy_ahp_method
```

Figura A.1: Tela de Instalação da Biblioteca.
Fonte: Adaptado de Valdecy [132].



```python
# Function: Fuzzy AHP
def fahp(dataset):
    row_sum = []
    s_row   = []
    f_w     = []
    d_w     = []
    #inc_rat = np.array([0, 0, 0, 0.58, 0.9, 1.12, 1.24, 1.32, 1.41, 1.45])
    inc_rat = np.array([0, 0, 0, 0.58, 0.9, 1.12, 1.24, 1.32, 1.41, 1.45, 1.49, 1.51, 1.48, 1.56, 1.57, 1.59]) #Novo índice de consistência randômico
    X       = [(item[0] + 4*item[1] + item[2])/6 for i in range(0, len(dataset)) for item in dataset[i] ]
    X       = np.asarray(X)
    X       = np.reshape(X, (len(dataset), len(dataset)))
    for i in range(0, len(dataset)):
        a, b, c = 1, 1, 1
        for j in range(0, len(dataset[i])):
            d, e, f = dataset[i][j]
            a, b, c = a*d, b*e, c*f
        row_sum.append( (a, b, c) )
    L, M, U = 0, 0, 0
    for i in range(0, len(row_sum)):
        a, b, c = row_sum[i]
        a, b, c = a**(1/len(dataset)), b**(1/len(dataset)), c**(1/len(dataset))
        s_row.append( ( a, b, c ) )
        L = L + a
        M = M + b
        U = U + c
    for i in range(0, len(s_row)):
        a, b, c = s_row[i]
        a, b, c = a*(U**-1), b*(M**-1), c*(L**-1)
        f_w.append( ( a, b, c ) )
        d_w.append( (a + b + c)/3 )
    n_w     = [item/sum(d_w) for item in d_w]
    vector  = np.sum(X*n_w, axis = 1)/n_w
    lamb_max = np.mean(vector)
    cons_ind = (lamb_max - X.shape[1])/(X.shape[1] - 1)
    rc      = cons_ind/inc_rat[X.shape[1]]
    return f_w, d_w, n_w, rc
```

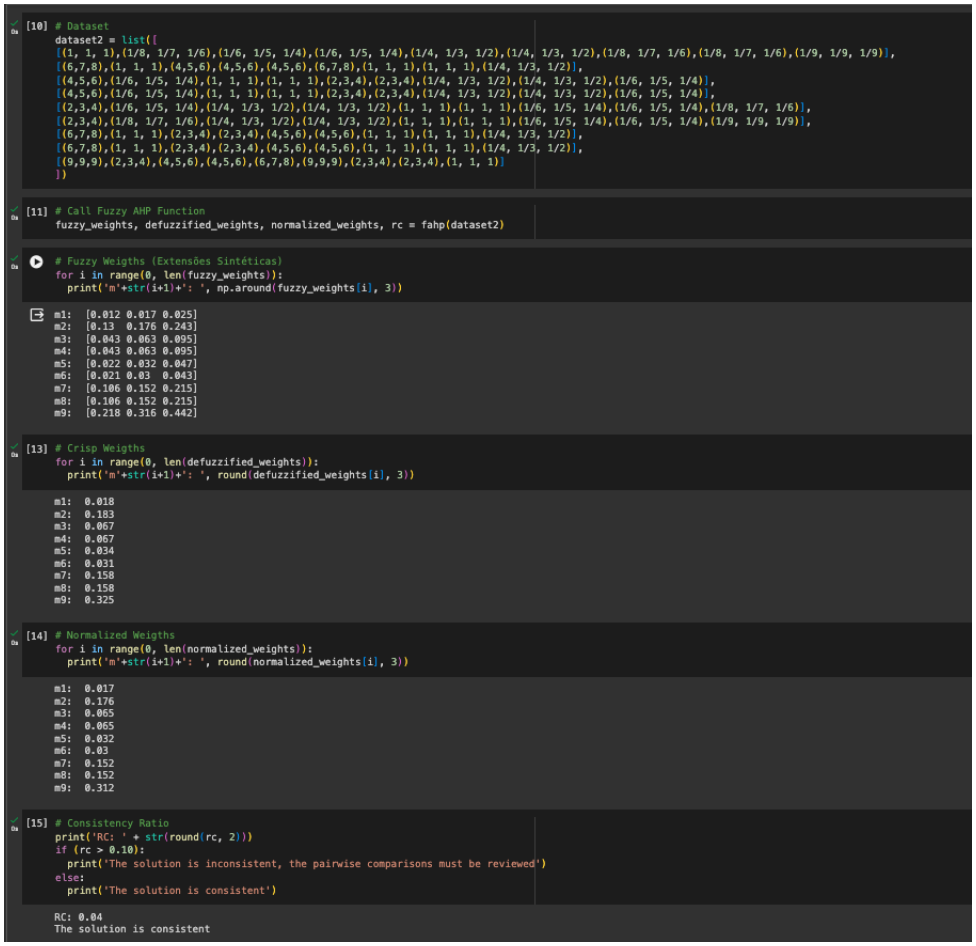
Figura A.2: Tela de Função.
Fonte: Adaptado de Valdecy [132].



Critério Pesos:

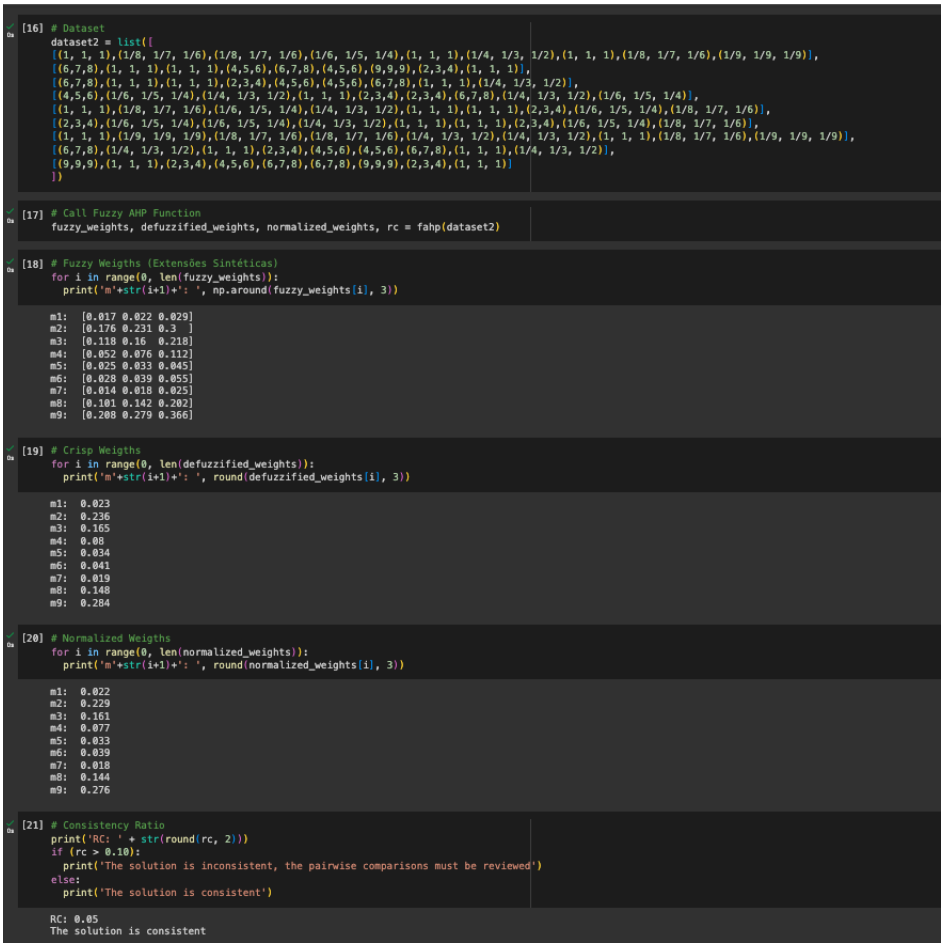
```python
[4] # Dataset
    dataset = list([
    [(1, 1, 1),(6,7,8),(2,3,4),(9,9,9),(4,5,6)],
    [(1/8, 1/7, 1/6),(1, 1, 1),(1/6, 1/5, 1/4),(2,3,4),(1/4, 1/3, 1/2)],
    [(1/4, 1/3, 1/2),(4,5,6),(1, 1, 1),(6,7,8),(2,3,4)],
    [(1/9, 1/9, 1/9),(1/4, 1/3, 1/2),(1/8, 1/7, 1/6),(1, 1, 1),(1/6, 1/5, 1/4)],
    [(1/6, 1/5, 1/4),(2,3,4),(1/4, 1/3, 1/2),(4,5,6),(1, 1, 1)],
    ])

[5] # Call Fuzzy AHP Function
    fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset)

    # Fuzzy Weigths (Extensões Sintéticas)
    for i in range(0, len(fuzzy_weights)):
      print('r'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

    r1:  [0.371 0.51  0.69 ]
    r2:  [0.044 0.064 0.094]
    r3:  [0.181 0.264 0.387]
    r4:  [0.025 0.033 0.046]
    r5:  [0.088 0.13  0.193]

[7] # Crisp Weigths
    for i in range(0, len(defuzzified_weights)):
      print('r'+str(i+1)+': ', round(defuzzified_weights[i], 3))

    r1:  0.523
    r2:  0.067
    r3:  0.277
    r4:  0.035
    r5:  0.137

    # Normalized Weigths
    for i in range(0, len(normalized_weights)):
      print('r'+str(i+1)+': ', round(normalized_weights[i], 3))

    r1:  0.503
    r2:  0.065
    r3:  0.267
    r4:  0.033
    r5:  0.132

[9] # Consistency Ratio
    print('RC: ' + str(round(rc, 2)))
    if (rc > 0.10):
      print('The solution is inconsistent, the pairwise comparisons must be reviewed')
    else:
      print('The solution is consistent')

    RC: 0.06
    The solution is consistent
```

Figura A.3: Tela Critério de Risco.
Fonte: Adaptado de Valdecy [132].



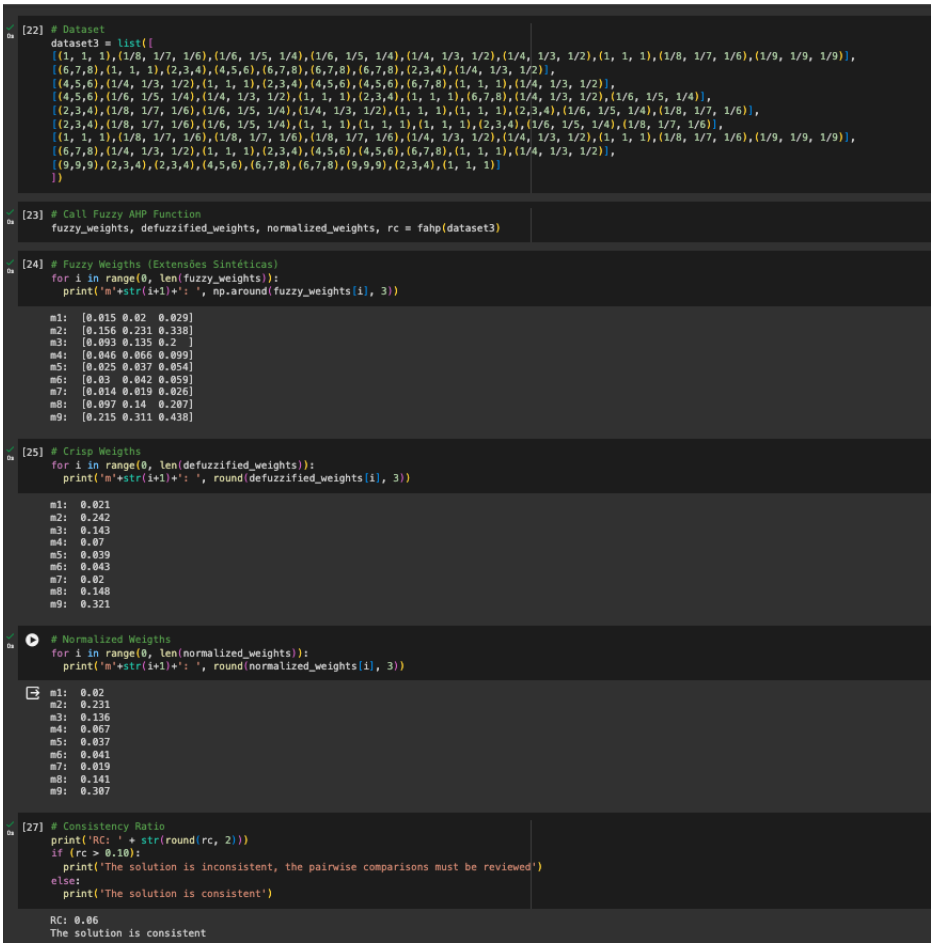
```
[10] # Dataset
     dataset2 = list([
     [(1, 1, 1),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/8, 1/7, 1/6),(1/8, 1/7, 1/6),(1/9, 1/9, 1/9)],
     [(6,7,8),(1, 1, 1),(4,5,6),(4,5,6),(4,5,6),(6,7,8),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
     [(4,5,6),(1/6, 1/5, 1/4),(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/6, 1/5, 1/4)],
     [(4,5,6),(1/6, 1/5, 1/4),(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/6, 1/5, 1/4)],
     [(2,3,4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6)],
     [(2,3,4),(1/8, 1/7, 1/6),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/9, 1/9, 1/9)],
     [(6,7,8),(1, 1, 1),(2,3,4),(2,3,4),(4,5,6),(4,5,6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
     [(6,7,8),(1, 1, 1),(2,3,4),(2,3,4),(4,5,6),(4,5,6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
     [(9,9,9),(2,3,4),(4,5,6),(4,5,6),(6,7,8),(9,9,9),(2,3,4),(2,3,4),(1, 1, 1)]
     ])

[11] # Call Fuzzy AHP Function
     fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset2)

     # Fuzzy Weigths (Extensões Sintéticas)
     for i in range(0, len(fuzzy_weights)):
       print('m'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

     m1:  [0.012 0.017 0.025]
     m2:  [0.13  0.176 0.243]
     m3:  [0.043 0.063 0.095]
     m4:  [0.043 0.063 0.095]
     m5:  [0.022 0.032 0.047]
     m6:  [0.021 0.03  0.043]
     m7:  [0.106 0.152 0.215]
     m8:  [0.106 0.152 0.215]
     m9:  [0.218 0.316 0.442]

[13] # Crisp Weigths
     for i in range(0, len(defuzzified_weights)):
       print('m'+str(i+1)+': ', round(defuzzified_weights[i], 3))

     m1:  0.018
     m2:  0.183
     m3:  0.067
     m4:  0.067
     m5:  0.034
     m6:  0.031
     m7:  0.158
     m8:  0.158
     m9:  0.325

[14] # Normalized Weigths
     for i in range(0, len(normalized_weights)):
       print('m'+str(i+1)+': ', round(normalized_weights[i], 3))

     m1:  0.017
     m2:  0.176
     m3:  0.065
     m4:  0.065
     m5:  0.032
     m6:  0.03
     m7:  0.152
     m8:  0.152
     m9:  0.312

[15] # Consistency Ratio
     print('RC: ' + str(round(rc, 2)))
     if (rc > 0.10):
       print('The solution is inconsistent, the pairwise comparisons must be reviewed')
     else:
       print('The solution is consistent')

     RC: 0.04
     The solution is consistent
```

Figura A.4: Tela de Vulnerabilidade por Módulo.
Fonte: Adaptado de Valdecy [132].



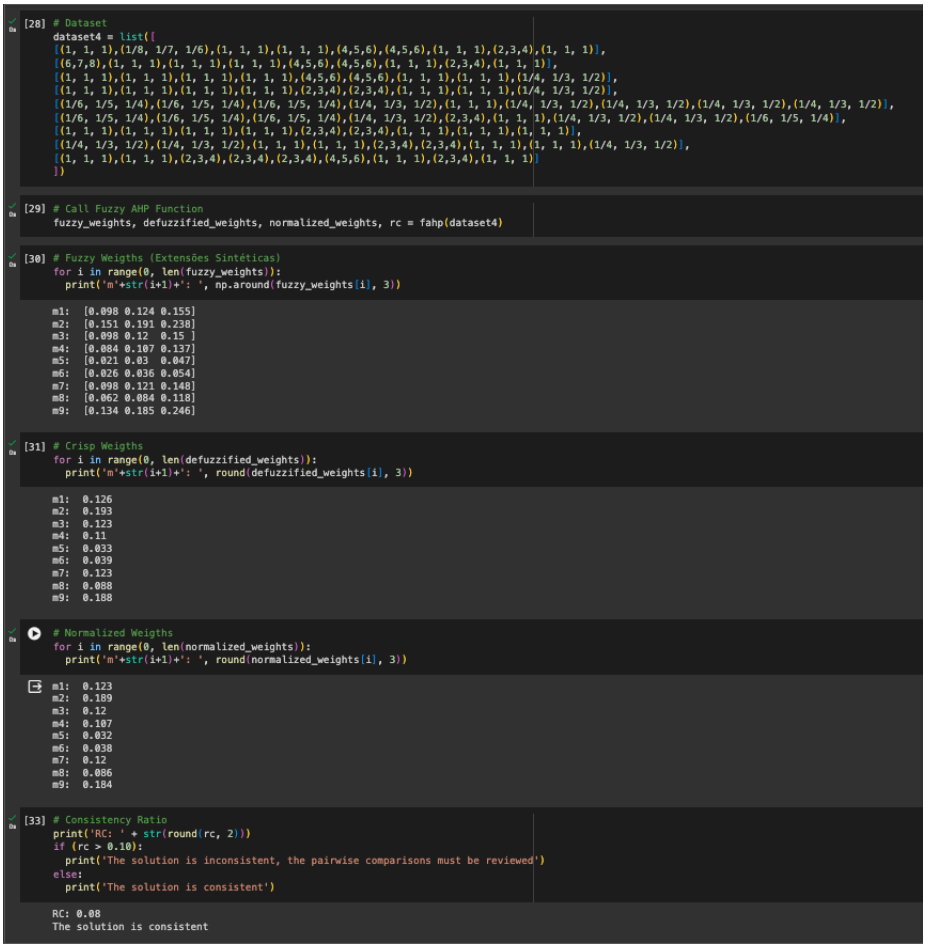
```
[16] # Dataset
dataset2 = list([
[(1, 1, 1),(1/8, 1/7, 1/6),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1, 1, 1),(1/4, 1/3, 1/2),(1, 1, 1),(1/8, 1/7, 1/6),(1/9, 1/9, 1/9)],
[(6,7,8),(1, 1, 1),(1, 1, 1),(4,5,6),(6,7,8),(4,5,6),(9,9,9),(2,3,4),(1, 1, 1)],
[(6,7,8),(1, 1, 1),(1, 1, 1),(2,3,4),(4,5,6),(4,5,6),(6,7,8),(1, 1, 1),(1/4, 1/3, 1/2)],
[(4,5,6),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(2,3,4),(2,3,4),(6,7,8),(1/4, 1/3, 1/2),(1/6, 1/5, 1/4)],
[(1, 1, 1),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(2,3,4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6)],
[(2,3,4),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(2,3,4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6)],
[(1, 1, 1),(1/9, 1/9, 1/9),(1/8, 1/7, 1/6),(1/8, 1/7, 1/6),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1/8, 1/7, 1/6),(1/9, 1/9, 1/9)],
[(6,7,8),(1/4, 1/3, 1/2),(1, 1, 1),(2,3,4),(4,5,6),(4,5,6),(6,7,8),(1, 1, 1),(1/4, 1/3, 1/2)],
[(9,9,9),(1, 1, 1),(2,3,4),(4,5,6),(6,7,8),(6,7,8),(9,9,9),(2,3,4),(1, 1, 1)]
])

[17] # Call Fuzzy AHP Function
fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset2)

[18] # Fuzzy Weigths (Extensões Sintéticas)
for i in range(0, len(fuzzy_weights)):
  print('m'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

m1:  [0.017 0.022 0.029]
m2:  [0.176 0.231 0.3  ]
m3:  [0.118 0.16  0.218]
m4:  [0.052 0.076 0.112]
m5:  [0.025 0.033 0.045]
m6:  [0.028 0.039 0.055]
m7:  [0.014 0.018 0.025]
m8:  [0.101 0.142 0.202]
m9:  [0.208 0.279 0.366]

[19] # Crisp Weigths
for i in range(0, len(defuzzified_weights)):
  print('m'+str(i+1)+': ', round(defuzzified_weights[i], 3))

m1:  0.023
m2:  0.236
m3:  0.165
m4:  0.08
m5:  0.034
m6:  0.041
m7:  0.019
m8:  0.148
m9:  0.284

[20] # Normalized Weigths
for i in range(0, len(normalized_weights)):
  print('m'+str(i+1)+': ', round(normalized_weights[i], 3))

m1:  0.022
m2:  0.229
m3:  0.161
m4:  0.077
m5:  0.033
m6:  0.039
m7:  0.018
m8:  0.144
m9:  0.276

[21] # Consistency Ratio
print('RC: ' + str(round(rc, 2)))
if (rc > 0.10):
  print('The solution is inconsistent, the pairwise comparisons must be reviewed')
else:
  print('The solution is consistent')

RC: 0.05
The solution is consistent
```

Figura A.5: Tela de Code Smell por Módulo.
Fonte: Adaptado de Valdecy [132].



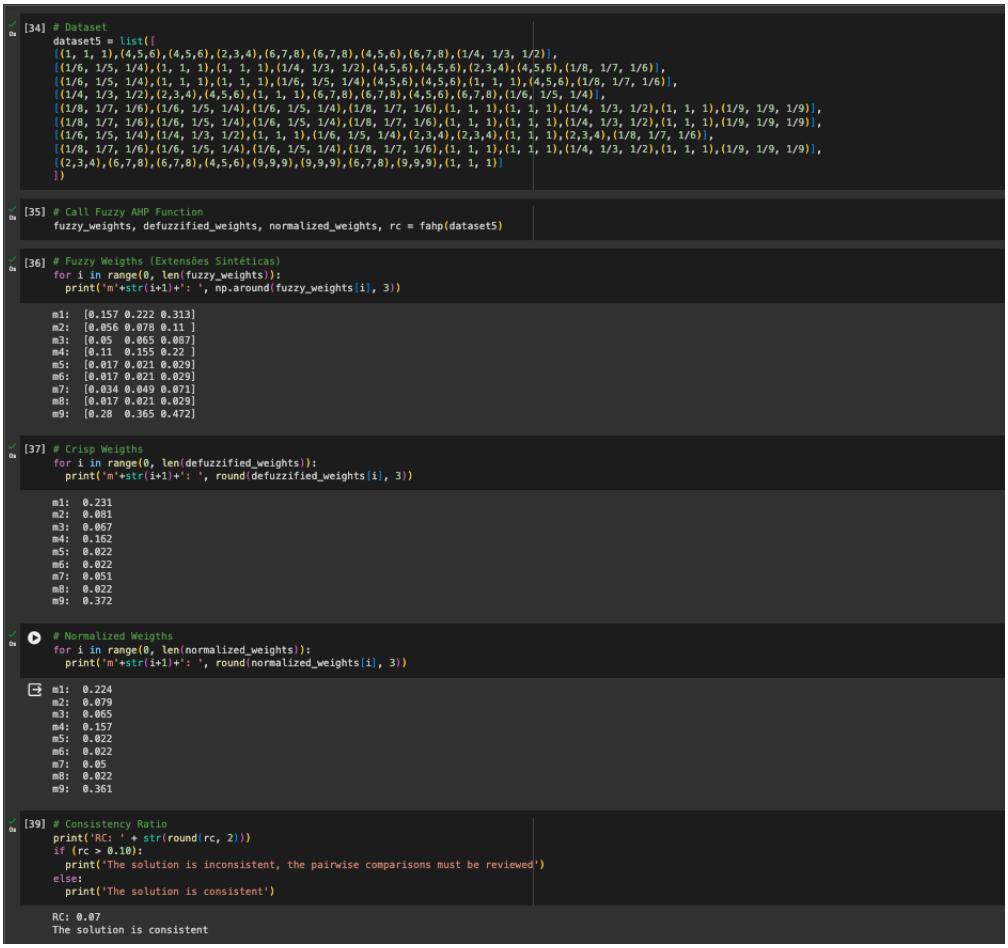
```
[22] # Dataset
dataset3 = list([
[(1, 1, 1),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1/8, 1/7, 1/6),(1/9, 1/9, 1/9)],
[(6,7,8),(1, 1, 1),(2,3,4),(4,5,6),(6,7,8),(6,7,8),(6,7,8),(2,3,4),(1/4, 1/3, 1/2)],
[(4,5,6),(1/4, 1/3, 1/2),(1, 1, 1),(2,3,4),(4,5,6),(4,5,6),(6,7,8),(1, 1, 1),(1/4, 1/3, 1/2)],
[(4,5,6),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(2,3,4),(1, 1, 1),(6,7,8),(1/4, 1/3, 1/2),(1/6, 1/5, 1/4)],
[(2,3,4),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(2,3,4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6)],
[(2,3,4),(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1, 1, 1),(1, 1, 1),(1, 1, 1),(2,3,4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6)],
[(1, 1, 1),(1/8, 1/7, 1/6),(1/8, 1/7, 1/6),(1/8, 1/7, 1/6),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1/8, 1/7, 1/6),(1/9, 1/9, 1/9)],
[(6,7,8),(1/4, 1/3, 1/2),(1, 1, 1),(2,3,4),(4,5,6),(4,5,6),(6,7,8),(1, 1, 1),(1/4, 1/3, 1/2)],
[(9,9,9),(2,3,4),(2,3,4),(4,5,6),(6,7,8),(6,7,8),(9,9,9),(2,3,4),(1, 1, 1)]
])

[23] # Call Fuzzy AHP Function
fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset3)

[24] # Fuzzy Weigths (Extensões Sintéticas)
for i in range(0, len(fuzzy_weights)):
  print('m'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

m1:  [0.015 0.02  0.029]
m2:  [0.156 0.231 0.338]
m3:  [0.093 0.135 0.2  ]
m4:  [0.046 0.066 0.099]
m5:  [0.025 0.037 0.054]
m6:  [0.03  0.042 0.059]
m7:  [0.014 0.019 0.026]
m8:  [0.097 0.14  0.207]
m9:  [0.215 0.311 0.438]

[25] # Crisp Weigths
for i in range(0, len(defuzzified_weights)):
  print('m'+str(i+1)+': ', round(defuzzified_weights[i], 3))

m1:  0.021
m2:  0.242
m3:  0.143
m4:  0.07
m5:  0.039
m6:  0.043
m7:  0.02
m8:  0.148
m9:  0.321

# Normalized Weigths
for i in range(0, len(normalized_weights)):
  print('m'+str(i+1)+': ', round(normalized_weights[i], 3))

m1:  0.02
m2:  0.231
m3:  0.136
m4:  0.067
m5:  0.037
m6:  0.041
m7:  0.019
m8:  0.141
m9:  0.307

[27] # Consistency Ratio
print('RC: ' + str(round(rc, 2)))
if (rc > 0.10):
  print('The solution is inconsistent, the pairwise comparisons must be reviewed')
else:
  print('The solution is consistent')

RC: 0.06
The solution is consistent
```

Figura A.6: Tela de Bugs por Módulo.
Fonte: Adaptado de Valdecy [132].



```
[28] # Dataset
dataset4 = list([
[(1, 1, 1),(1/8, 1/7, 1/6),(1, 1, 1),(1, 1, 1),(4,5,6),(4,5,6),(1, 1, 1),(2,3,4),(1, 1, 1)],
[(6,7,8),(1, 1, 1),(1, 1, 1),(1, 1, 1),(4,5,6),(4,5,6),(1, 1, 1),(2,3,4),(1, 1, 1)],
[(1, 1, 1),(1, 1, 1),(1, 1, 1),(1, 1, 1),(4,5,6),(4,5,6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
[(1, 1, 1),(1, 1, 1),(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
[(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2)],
[(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(2,3,4),(1, 1, 1),(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1/6, 1/5, 1/4)],
[(1, 1, 1),(1, 1, 1),(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(1, 1, 1),(1, 1, 1),(1, 1, 1)],
[(1/4, 1/3, 1/2),(1/4, 1/3, 1/2),(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2)],
[(1, 1, 1),(1, 1, 1),(2,3,4),(2,3,4),(2,3,4),(4,5,6),(1, 1, 1),(2,3,4),(1, 1, 1)]
])

[29] # Call Fuzzy AHP Function
fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset4)

[30] # Fuzzy Weigths (Extensões Sintéticas)
for i in range(0, len(fuzzy_weights)):
  print('m'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

m1:  [0.098 0.124 0.155]
m2:  [0.151 0.191 0.238]
m3:  [0.098 0.12  0.15 ]
m4:  [0.084 0.107 0.137]
m5:  [0.021 0.03  0.047]
m6:  [0.026 0.036 0.054]
m7:  [0.098 0.121 0.148]
m8:  [0.062 0.084 0.118]
m9:  [0.134 0.185 0.246]

[31] # Crisp Weigths
for i in range(0, len(defuzzified_weights)):
  print('m'+str(i+1)+': ', round(defuzzified_weights[i], 3))

m1:  0.126
m2:  0.193
m3:  0.123
m4:  0.11
m5:  0.033
m6:  0.039
m7:  0.123
m8:  0.088
m9:  0.188

# Normalized Weigths
for i in range(0, len(normalized_weights)):
  print('m'+str(i+1)+': ', round(normalized_weights[i], 3))

m1:  0.123
m2:  0.189
m3:  0.12
m4:  0.107
m5:  0.032
m6:  0.038
m7:  0.12
m8:  0.086
m9:  0.184

[33] # Consistency Ratio
print('RC: ' + str(round(rc, 2)))
if (rc > 0.10):
  print('The solution is inconsistent, the pairwise comparisons must be reviewed')
else:
  print('The solution is consistent')

RC: 0.08
The solution is consistent
```

Figura A.7: Tela de Plágio/Similaridade por Módulo.
Fonte: Adaptado de Valdecy [132].



```
[34] # Dataset
dataset5 = list([
[(1, 1, 1),(4,5,6),(4,5,6),(2,3,4),(6,7,8),(6,7,8),(4,5,6),(6,7,8),(1/4, 1/3, 1/2)],
[(1/6, 1/5, 1/4),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2),(4,5,6),(4,5,6),(2,3,4),(4,5,6),(1/8, 1/7, 1/6)],
[(1/6, 1/5, 1/4),(1, 1, 1),(1, 1, 1),(1/6, 1/5, 1/4),(4,5,6),(4,5,6),(1, 1, 1),(4,5,6),(1/8, 1/7, 1/6)],
[(1/4, 1/3, 1/2),(2,3,4),(4,5,6),(1, 1, 1),(6,7,8),(6,7,8),(4,5,6),(6,7,8),(1/6, 1/5, 1/4)],
[(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2),(1, 1, 1),(1/9, 1/9, 1/9)],
[(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2),(1, 1, 1),(1/9, 1/9, 1/9)],
[(1/6, 1/5, 1/4),(1/4, 1/3, 1/2),(1, 1, 1),(1/6, 1/5, 1/4),(2,3,4),(2,3,4),(1, 1, 1),(2,3,4),(1/8, 1/7, 1/6)],
[(1/8, 1/7, 1/6),(1/6, 1/5, 1/4),(1/6, 1/5, 1/4),(1/8, 1/7, 1/6),(1, 1, 1),(1, 1, 1),(1/4, 1/3, 1/2),(1, 1, 1),(1/9, 1/9, 1/9)],
[(2,3,4),(6,7,8),(6,7,8),(4,5,6),(9,9,9),(9,9,9),(6,7,8),(9,9,9),(1, 1, 1)]
])

[35] # Call Fuzzy AHP Function
fuzzy_weights, defuzzified_weights, normalized_weights, rc = fahp(dataset5)

[36] # Fuzzy Weigths (Extensões Sintéticas)
for i in range(0, len(fuzzy_weights)):
  print('m'+str(i+1)+': ', np.around(fuzzy_weights[i], 3))

m1:  [0.157 0.222 0.313]
m2:  [0.056 0.078 0.11 ]
m3:  [0.05  0.065 0.087]
m4:  [0.11  0.155 0.22 ]
m5:  [0.017 0.021 0.029]
m6:  [0.017 0.021 0.029]
m7:  [0.034 0.049 0.071]
m8:  [0.017 0.021 0.029]
m9:  [0.28  0.365 0.472]

[37] # Crisp Weigths
for i in range(0, len(defuzzified_weights)):
  print('m'+str(i+1)+': ', round(defuzzified_weights[i], 3))

m1:  0.231
m2:  0.081
m3:  0.067
m4:  0.162
m5:  0.022
m6:  0.022
m7:  0.051
m8:  0.022
m9:  0.372

# Normalized Weigths
for i in range(0, len(normalized_weights)):
  print('m'+str(i+1)+': ', round(normalized_weights[i], 3))

m1:  0.224
m2:  0.079
m3:  0.065
m4:  0.157
m5:  0.022
m6:  0.022
m7:  0.05
m8:  0.022
m9:  0.361

[39] # Consistency Ratio
print('RC: ' + str(round(rc, 2)))
if (rc > 0.10):
  print('The solution is inconsistent, the pairwise comparisons must be reviewed')
else:
  print('The solution is consistent')

RC: 0.07
The solution is consistent
```

Figura A.8: Tela de Uso Indevido de Licença por Módulo.
Fonte: Adaptado de Valdecy [132].