Aurora View: Uma abordagem de coleta, unificação e visualização de dados de dívida técnica

Gabriel Soares Santos[1]

[1]Centro de Informática – Universidade Federal da Paraíba (UFPB)
João Pessoa – PB – Brasil

gabriel.soares@academico.ufpb.br

Abstract. *This work proposes Aurora View, an approach for automated collection, unification, and visualization of technical debt metrics in software projects. By integrating version control systems and code quality analysis platforms, the system aims to simplify monitoring infrastructure and maximize applicability in real software development contexts. The research adopts Design Science Research (DSR) methodology to develop a computational artifact that supports managerial decisions about refactoring prioritization and code quality management throughout the project lifecycle. Results show the system successfully integrated data from multiple sources, providing unified dashboards adapted for different stakeholders with focus on the leak period concept for preventive technical debt management. The validation demonstrates effectiveness in synchronizing real-time data between platforms, processing large volumes of repository information, and unifying previously fragmented metrics. The approach contributes to bridging the gap between technical debt theory and practical industrial application through empirically validated visualization patterns.*

Resumo. *Este trabalho propõe o Aurora View, uma abordagem para coleta, unificação e visualização automatizada de métricas de dívida técnica de código em projetos de software. Utilizando a integração entre sistemas de controle de versão e plataformas de análise de qualidade de código, o sistema visa simplificar a infraestrutura de monitoramento e maximizar a aplicabilidade em contextos reais de projetos de software. A pesquisa adota a metodologia Design Science Research (DSR) para desenvolver um artefato computacional que apoie decisões gerenciais sobre priorização de refatoração e gestão da qualidade de código ao longo do ciclo de vida do projeto. Os resultados obtidos demonstram que o sistema conseguiu integrar dados de múltiplas fontes, fornecendo dashboards unificados adaptados para diferentes stakeholders com foco no conceito de leak period para gestão preventiva da dívida técnica. A validação demonstra eficácia na sincronização de dados em tempo real entre plataformas, processamento de grandes volumes de informações de repositórios e unificação de métricas anteriormente fragmentadas. A abordagem contribui para preencher a lacuna entre teoria de dívida técnica e aplicação prática industrial através de padrões de visualização empiricamente validados.*

1. Introducao

A crescente complexidade dos ambientes de desenvolvimento de software modernos tem evidenciado a necessidade crítica de monitoramento contínuo da qualidade do código

e gestão eficaz da dívida técnica, conceito que evoluiu significativamente desde sua concepção original por Ward Cunningham em 1992 através de seu trabalho seminal ”The WyCash Portfolio Management System” [6]. O que começou como uma metáfora comunicacional simples para explicar compensações técnicas a partes interessadas não-técnicas transformou-se em um *framework* estratégico sofisticado que representa o custo futuro de retrabalho causado pela escolha de soluções de curto prazo ao invés de abordagens mais robustas que demandariam maior tempo inicial [2]. As revisões sistemáticas contemporâneas, incluindo o abrangente trabalho de Behutiye et al. (2024) que analisou 38 estudos primários sobre dívida técnica em desenvolvimento ágil, demonstram que o conceito se expandiu para múltiplas dimensões, abrangendo desde dívida de código e arquitetura até dívida de documentação, processos e requisitos, estabelecendo-se como um dos principais desafios na engenharia de software moderna [5]. A recente síntese de Ozkaya e Shull (2024) sobre o futuro da gestão de dívida técnica confirma que esta área emergiu como disciplina fundamental, com mais de 532 estudos primários únicos identificados em revisões terciárias, indicando maturidade científica crescente [24].

Em contextos empresariais contemporâneos, onde a pressão por entregas rápidas e ciclos de desenvolvimento acelerados é constante, o acúmulo descontrolado de dívida técnica pode comprometer significativamente a sustentabilidade, evolutibilidade e saúde operacional dos sistemas a longo prazo. Estudos empíricos recentes revelam que organizações com alta carga de dívida técnica experimentam degradação significativa na velocidade de entrega, aumento nas taxas de falha de *deployments* e redução na capacidade de resposta às mudanças de mercado, com impactos financeiros que podem superar 40-50% dos custos operacionais previstos [18]. A fragmentação de dados entre diferentes ferramentas de desenvolvimento - incluindo sistemas de controle de versão como GitHub e GitLab, plataformas de integração contínua como Jenkins e Azure DevOps, ferramentas de análise de qualidade como SonarQube e SonarCloud, e gerenciadores de projeto como Jira e ClickUp - cria silos informacionais que dificultam substancialmente a obtenção de uma visão unificada e holística sobre o estado real da dívida técnica [19]. Esta dispersão de informações impede correlações efetivas entre indicadores de diferentes domínios, obscurecendo padrões críticos que poderiam sinalizar problemas emergentes ou oportunidades de otimização no processo de desenvolvimento, conforme evidenciado pelo mapeamento sistemático de Santos et al. (2024) sobre práticas de monitoramento de dívida técnica [27].

A dívida técnica transcende significativamente a dimensão meramente técnica, configurando-se como um desafio fundamental de negócio com impactos diretos no desempenho organizacional e nos resultados financeiros das empresas desenvolvedoras de software. Pesquisas empíricas recentes, incluindo o estudo longitudinal de Freire et al. (2023) com mais de 200 profissionais da indústria, indicam que organizações com alta carga de dívida técnica experimentam degradação significativa na velocidade de entrega, com reduções médias de 30-40% na produtividade das equipes de desenvolvimento [10]. A inércia na gestão proativa da dívida técnica pode resultar em falhas sistêmicas de projetos, criando um ciclo vicioso de deterioração progressiva da qualidade e produtividade que se manifesta através de aumento exponencial nos tempos de desenvolvimento de novas funcionalidades, maior frequência de *bugs* em produção, e custos crescentes de manutenção [8]. O estudo de Kleinwaks et al. (2023) sobre dívida técnica em engenharia de sistemas revela que, embora o conceito seja prevalente na engenharia de soft-

ware, sua aplicação em domínios mais amplos ainda carece de taxonomias comuns e *frameworks* sistemáticos [16], indicando oportunidades significativas para contribuições metodológicas como a proposta neste trabalho.

A natureza frequentemente invisível da dívida técnica para tomadores de decisão não-técnicos cria uma situação particularmente perigosa, onde riscos e custos se acumulam de forma silenciosa até atingirem níveis críticos que comprometem a viabilidade dos sistemas. A ausência de métricas objetivas e visualizações compreensíveis impede que gestores e partes interessadas compreendam adequadamente o impacto da dívida técnica nos cronogramas, orçamentos e qualidade das entregas, problema agravado pela falta de ferramentas práticas para priorização sistemática identificada por Lenarduzzi et al. (2021) em sua revisão de 557 *papers* únicos [19]. Transformar esta questão tradicionalmente subjetiva em uma análise objetiva, quantitativa e orientada por dados torna-se fundamental para a sustentabilidade dos projetos de software modernos, especialmente considerando que estudos recentes mostram que até 25% do tempo de desenvolvimento pode ser consumido por atividades relacionadas à dívida técnica [11]. A pesquisa de Maciel et al. (2024) sobre gestão de dívida técnica em metodologias ágeis confirma que, embora Scrum seja a metodologia mais utilizada para gestão da dívida técnica, ainda existem lacunas significativas na integração de ferramentas e métricas automatizadas [21], justificando a necessidade de soluções como o Aurora View.

As bases teóricas para medição e monitoramento de dívida técnica, estabelecidas por pesquisadores pioneiros na área e consolidadas no Seminário Dagstuhl 2016, definem um *framework* de três estágios fundamentais: identificação sistemática de itens de dívida técnica através de análise estática e dinâmica de código, quantificação precisa do esforço necessário para sua eliminação baseada em métricas como o *Technical Debt Ratio* (TDR), e implementação de estratégias de monitoramento contínuo para determinar o momento ótimo de intervenções corretivas [2]. Este *framework* teórico tem evoluído para incorporar métricas sofisticadas como índices de qualidade baseados em ciclo de vida de software (SQALE), análises comportamentais de código que correlacionam padrões de desenvolvimento com acúmulo de dívida técnica, e abordagens preventivas baseadas no conceito de "*leak period*" que foca exclusivamente em código novo para evitar acúmulo adicional de dívida [29]. A integração dessas métricas com ferramentas de DevOps modernas, conforme demonstrado por práticas de mercado sobre a integração contínua entre sistemas de versionamento e ferramentas de análise de qualidade, representa uma tendência crescente em direção à automação e integração contínua na gestão de qualidade de código [1].

O objetivo central deste estudo consiste em desenvolver e avaliar empiricamente o Aurora View, um sistema abrangente de coleta automatizada, unificação inteligente e visualização adaptativa de métricas de dívida técnica que integre dados provenientes de plataformas heterogêneas de versionamento e monitoramento de qualidade de código em *dashboards* contextualizados para diferentes perfis de usuário, incluindo desenvolvedores, gestores técnicos e partes interessadas de negócio. A solução proposta visa democratizar o acesso às métricas críticas de qualidade de software, transformando informações técnicas complexas em visualizações compreensíveis e acionáveis que facilitem a tomada de decisão informada em todos os níveis organizacionais, preenchendo assim a lacuna identificada por múltiplos estudos entre ferramentas teóricas de análise de dívida técnica e sua aplicação prática na indústria [24]. Utilizando a metodologia *Design Science Research*

(DSR) conforme estabelecida por Hevner et al. (2004) para sistemas de informação e posteriormente especializada por Wieringa (2014) para engenharia de software, o estudo contribui tanto para o avanço teórico da área quanto para a prática industrial através de um artefato computacional empiricamente validado [13, 32].

A complexidade crescente dos ambientes virtuais modernos de desenvolvimento de software, impulsinada pela adoção de práticas DevOps, trouxe uma proliferação de ferramentas especializadas que cobrem o ciclo de vida e entrega de software. Desde o controle de versão(e.g., Github, GitLab) e gerenciamento de projeto(e.g., Jira, ClickUp) até a Integração e Entrega contínua(CI/CD(e.g., Jenkins)) e análise de código (e.g., SonarQube), as equipes operam em um ecossistema que gera e disponibiliza muitos dados, mas normalmente fragmentado. Essa dispersão de informações representa um desafio para o monitoramento holístico da saúde(*health checking*) dos processos DevOps, dificultando a obtenção de uma visão unificada, a identificação de gargalos, sinais de alerta ou a correlação entre identificadores de diferentes domínios.

A ausência de uma visão consolidada não apenas compromete a tomada de decisão ágil e informada, mas também pode mascarar problemas que afetam a sustentabilidade e a evolutibilidade dos sistemas a longo prazo. Entre esses problemas, destaca-se a Dívida Técnica Entrelaçada (DTE), caracterizada pela complexificação implícita da arquitetura e pelo surgimento de acoplamentos não intencionais, frequentemente exacerbada pela natureza iterativa e focada no usuário do DevOps (Bonet Faus et al., 2023).Essa problemática se estende e se agrava em domínios emergentes como o de Operações de (*Machine Learning* (MLOps), onde, conforme apontado por Karamitsos et al. (2020), a implantação e manutenção de modelos de ML, se não cuidadosamente projetadas, podem levar a abordagens complexas e demoradas, exigindo esforços significativos e custosos, e muitas vezes ocultando uma 'dívida técnica' específica aos sistemas. A necessidade de minimizar desperdício, suportar ciclos rápidos de feedback e melhorar a entrega de valor é, portanto, um desafio transversal no universo de DevOps.

Neste contexto, o presente trabalho propõe o Aurora View, uma abordagem e um artefato computacional projetados para a coleta, unificação e visualização de dados provenientes de múltiplas plataformas DevOps. Utilizando a metodologia Design Science Research (DSR), o Aurora View objetiva atacar o problema da fragmentação de dados importantes para o monitoramento do desenvolvimento e manutenção de sistemas. Diante disso, já com dados unificados, propor dashboards visuais que consolidam indicadores chave de saúde (KPIs) de fluxos de processos. O objetivo central é prover uma visão unificada e facilitar a análise correlacional, capacitando gestores e equipes a obterem insights claros e acionáveis sobre a saúde de seus processos, apoiando a tomada de decisão informada e contribuindo para a identificação de tendências que possam sinalizar proble, incluindo aqueles relacionados à dívida técnica. Este artigo detalha a motivação, os objetivos da solução, o design e desenvolvimento do artefato Aurora View, bem como a estratégia de validação de sua utilidade.

2. Referencial Teórico

2.1. Dívida Técnica: Conceitos Fundamentais e Evolução

A Dívida Técnica foi introduzida por Ward Cunningham em 1992 por meio de seu trabalho The WyCash Portfolio Management System, onde estabeleceu a metáfora fundamental (tradução livre): "Entregar um código pela primeira vez é como contrair uma dívida. Um pouco de dívida acelera o desenvolvimento, desde que seja paga prontamente com uma reescrita... O perigo acontece quando a dívida não é paga." [6]. Essa metáfora original tinha como objetivo explicar decisões de compensação às partes interessadas não técnicas, mas ao longo do tempo evoluiu para se tornar um referencial conceitual sólido na engenharia de software moderna. O Seminário Dagstuhl de 2016 sobre Managing Technical Debt in Software Engineering estabeleceu uma definição formal consolidada: Dívida Técnica é um conjunto de construções de projeto ou implementação convenientes no curto prazo que criam um contexto técnico custoso para mudanças futuras [2]. Essa definição enfatiza que a Dívida Técnica não é necessariamente resultado de má engenharia, mas pode surgir de escolhas conscientes e estratégicas baseadas em compensações, como demonstrado por Kruchten (2020) em sua análise das múltiplas dimensões desse conceito [18].

A evolução da noção de Dívida Técnica tem sido documentada em diversas revisões sistemáticas e estudos terciários. Behutiye et al. (2024) realizaram uma revisão sistemática voltada especificamente ao desenvolvimento ágil, analisando 38 estudos primários e identificando cinco áreas principais de pesquisa, com a gestão da Dívida Técnica em contextos ágeis recebendo maior destaque, seguida pela arquitetura e sua relação com o tema [5]. Já a síntese de Ozkaya e Shull (2024) mostra que, desde o primeiro workshop Managing Technical Debt em 2010, centenas de publicações surgiram. Estudos terciários recentes resumem resultados de 19 estudos secundários que, no total, analisaram 532 pesquisas primárias distintas [24]. Esse progresso evidencia não apenas a crescente maturidade científica da área, mas também a necessidade prática de soluções como o Aurora View, que buscam transformar o conhecimento teórico em ferramentas aplicáveis.

A taxonomia atual da Dívida Técnica abrange várias dimensões que vão além do código-fonte. Fagundes et al. (2023), em sua revisão sistemática sobre Dívida Técnica de requisitos, mostraram que esse tipo específico pode ser incorrido mesmo sem estar associado a problemas no código, revelando características próprias e impactos que podem se espalhar para outras atividades de desenvolvimento [9]. Yang et al. (2023), por sua vez, ampliaram o conceito para sistemas complexos, propondo uma taxonomia da Dívida Técnica relacionada a decisões envolvendo Commercial Off-The-Shelf (COTS), que inclui riscos de integração e obsolescência como formas específicas dessa dívida [33]. Essa diversificação conceitual reforça a relevância de abordagens unificadas de monitoramento, como a proposta pelo Aurora View.

2.2. Medição e Monitoramento de Dívida Técnica

As métricas fundamentais para quantificação de dívida técnica foram estabelecidas através de décadas de pesquisa empírica e industrial. Letouzey (2012) introduziu o método SQALE (Software Quality Assessment based on Lifecycle Expectations) que estabeleceu o Technical Debt Ratio (TDR) como métrica fundamental, definida como a

razão entre o custo de remediação e o custo de desenvolvimento por linha de código multiplicado pelo número de linhas [20]. Esta métrica traduz a dívida para termos econômicos, facilitando a comunicação com a gerência sobre a necessidade de alocar recursos, e foi posteriormente implementada como base de modelos de qualidade em ferramentas líderes de mercado como SonarQube/SonarCloud [3].

Avgeriou et al. (2020) conduziram uma análise comparativa detalhada de ferramentas comerciais e acadêmicas para medição de dívida técnica, destacando ferramentas como o SonarQube/SonarCloud como referência por sua abordagem baseada em SQALE e capacidades de integração com plataformas DevOps [3]. O estudo avaliou múltiplas ferramentas incluindo CAST, NDepend, e outras soluções proprietárias, mas confirmou a supremacia do ecossistema Sonar em termos de adoção industrial e robustez metodológica. Esta validação empírica justifica a escolha do Aurora View de se integrar com plataformas de análise de qualidade robustas, como o SonarCloud, para a coleta de métricas.

O conceito de ”leak period” emergiu como abordagem fundamental para gestão preventiva de dívida técnica, conforme documentado nas práticas oficiais da SonarSource [29]. Esta abordagem foca exclusivamente em código novo, permitindo que equipes mantenham qualidade em novas funcionalidades enquanto planejam estratégias de longo prazo para código legado, evitando o problema comum de teams ficarem sobrecarregadas pela dívida histórica. Almog (2023) demonstrou a implementação prática deste conceito através da integração de pipelines de CI/CD (como GitHub Actions) com plataformas de análise (como o SonarCloud), mostrando como automação de CI/CD pode incorporar verificações de *leak period* em *workflows* de desenvolvimento [1].

Santos et al. (2024) realizaram um mapeamento sistemático específico sobre práticas de monitoramento de dívida técnica, revelando que profissionais da indústria enfrentam desafios significativos na implementação de monitoramento sistemático, com 60% dos participantes relatando dificuldades na integração de ferramentas e 45% citando falta de métricas apropriadas [27]. Este *gap* entre necessidade prática e disponibilidade de soluções valida a relevância do Aurora View como contribuição para a área.

2.3. Code Smells e Indicadores de Qualidade

Code smells representam manifestações diretas da dívida técnica e constituem foco central para ferramentas de análise estática, conforme estabelecido por múltiplos estudos empíricos. Ramirez Lahti et al. (2021) conduziram estudo empírico sobre gestão de dívida técnica através de code smells e antipatterns, demonstrando como análise estática automatizada combinada com inspeção manual efetivamente detecta padrões problemáticos [26]. Os principais code smells correlacionados com TD incluem God Class, Long Method, Duplicate Code, Data Class e Feature Envy, cada um contribuindo de forma diferente para o acúmulo de dívida técnica.

A quantificação objetiva de duplicação de código estabelece que não mais que 5% do código deve ser duplicado em projetos de qualidade, conforme recomendações consolidadas na literatura [20]. Complexidade ciclomática, medindo o número de caminhos independentes através do código, deve idealmente permanecer abaixo de 10 por função para manter manutenibilidade adequada [11]. Cobertura de testes, fundamental para permitir refatoração segura, deve superar 80% para garantir detecção adequada de regressões

durante atividades de pagamento de dívida técnica.

Maldonado et al. (2020) desenvolveram abordagem específica para medição de TD usando SonarQube, focando em estudos de caso industriais que demonstraram correlação direta entre métricas automatizadas e percepção subjetiva de qualidade por desenvolvedores [22]. Este trabalho validou empiricamente que ferramentas de análise estática podem efetivamente quantificar aspectos qualitativos da dívida técnica, fornecendo base científica para abordagens automatizadas como a implementada no Aurora View.

2.4. Visualização de Dados de Software e Dashboard Design

A pesquisa em visualização de software tem evoluído significativamente, com conferências especializadas como VISSOFT (IEEE Working Conference on Software Visualization) fornecendo venues dedicados para validação de abordagens inovadoras em apresentação visual de métricas de software [31]. Bach et al. (2022) estabeleceram taxonomy abrangente de dashboard design patterns através de análise sistemática, identificando oito grupos fundamentais de padrões para composição e layout que podem ser aplicados especificamente para visualização de métricas de engenharia de software [4].

Kovalenko et al. (2018) desenvolveram ferramenta específica para visualização de dívida técnica visando melhorar comunicação entre stakeholders técnicos e não-técnicos [17]. Seu estudo de caso em ambiente industrial multi-nacional demonstrou que visualizações adaptadas por perfil de usuário servem como plataforma de comunicação eficaz entre diferentes níveis hierárquicos, validando empiricamente a abordagem de interfaces diferenciadas implementada no Aurora View. O estudo confirmou que executivos requerem métricas agregadas de alto nível, enquanto desenvolvedores necessitam detalhes técnicos específicos para ação imediata.

Pesquisas recentes sobre tendências em design de dashboards, documentadas por especialistas da indústria, indicam movimento toward visualizações mais limpas com foco em ”colorblocking” para destacar insights críticos, wireframing para organização consistente, e eliminação de elementos decorativos em favor de funcionalidade [23]. Estas tendências são particularmente relevantes para dashboards de engenharia de software, onde clareza e acionabilidade são prioritárias sobre apelo visual.

2.5. Integração DevOps e Ferramentas de Desenvolvimento

A integração de ferramentas DevOps para monitoramento contínuo de qualidade representa tendência crescente na indústria, documentada através de múltiplos estudos e tutoriais práticos. A documentação oficial da SonarSource detalha integrações nativas com GitHub, Azure DevOps, BitBucket e GitLab, demonstrando maturidade técnica dessas integrações [29]. Almog (2023) forneceu guia prático detalhado para a integração entre sistemas de CI/CD, como o GitHub Actions, e plataformas de análise de qualidade, como o SonarCloud, mostrando como desenvolvedores podem incorporar análise automatizada de qualidade em workflows de CI/CD [1].

Senapathi et al. (2019) documentaram implementação industrial de pipeline integrado usando GoCD-GitHub-Datadog, demonstrando aumento significativo de 30 para 120 releases mensais através de automação e monitoramento integrado [28]. Embora focado em métricas de entrega rather than qualidade de código, este estudo demonstra

viabilidade técnica e benefícios organizacionais de integrações automatizadas entre ferramentas DevOps.

A pesquisa sobre DevOps capabilities de Erich et al. (2019) através de estudo de caso revelou que organizações com integração efetiva de ferramentas experimentam melhorias significativas em lead time, deployment frequency e mean time to recovery [7]. Kato e Ishikawa (2024) propuseram framework baseado em ISO 25000 (SQuaRE) para captura automatizada de métricas de qualidade através do pipeline CI/CD, demonstrando convergência entre padrões de qualidade e práticas DevOps [15].

3. Metodologia

Este trabalho adota a metodologia *Design Science Research* (DSR) para desenvolver e avaliar o Aurora View como solução para o problema de monitoramento e visualização de dívida técnica. A DSR é uma metodologia de pesquisa reconhecida como apropriada para criar e avaliar artefatos tecnológicos que visam resolver problemas práticos do mundo real, conforme estabelecido pelos trabalhos fundadores de Hevner, March, Park e Ram (2004) que criaram o *framework* original de *Design Science Research* para sistemas de informação, e posteriormente especializado para engenharia de *software* por Wieringa (2014) [13, 32].

3.1. Design Science Research em Engenharia de Software

O *framework* DSR estabelecido pelos pesquisadores fundadores Hevner, March, Park e Ram (2004) na MIS *Quarterly* define critérios rigorosos para pesquisa em sistemas de informação, especificando que artefatos devem ser relevantes para problemas práticos, rigorosos em sua fundamentação teórica, e demonstrar utilidade através de avaliação empírica [13]. Peffers et al. (2007) especializaram essa abordagem propondo um processo estruturado de seis etapas: identificação do problema, definição de objetivos, *design* e desenvolvimento, demonstração, avaliação e comunicação [25].

Storey et al. (2020) analisaram especificamente como pesquisa em engenharia de *software* se alinha com *design science*, identificando que validação deve combinar demonstração técnica com avaliação empírica através de estudos de caso e experimentos controlados [30]. Para validação de artefatos computacionais em engenharia de *software*, Wieringa (2014) propõe *Technical Action Research* como método específico que combina desenvolvimento iterativo com validação em contexto organizacional real [32]. Esta abordagem é particularmente adequada para o Aurora View por permitir validação em ambiente de desenvolvimento real.

O *framework* de avaliação de artefatos estabelecido pela conferência ICSE define critérios claros: *Available* (publicamente acessível), *Functional* (funciona conforme especificado), *Reusable* (pode ser adaptado para outros contextos), *Replicated* (outros pesquisadores podem executar), e *Reproduced* (resultados podem ser verificados independentemente) [14]. Estes critérios guiam tanto o desenvolvimento quanto a validação do Aurora View.

3.2. Processo de Pesquisa Aplicado

3.2.1. Identificação do Problema e Motivação

A identificação do problema foi baseada em revisão rápida da literatura e análise das necessidades práticas de organizações de desenvolvimento de software. Foi realizada busca estruturada nas bases ACM Digital Library, IEEE Xplore, Springer Link, e ArXiv, iniciando com 346 artigos e resultando na análise de 41 referências de alta qualidade baseadas em critérios de relevância, atualidade (2020-2024), e 8 artigos relacionados.

O problema central identificado foi a fragmentação de dados de dívida técnica entre diferentes ferramentas (sistemas de controle de versão, ferramentas de análise de qualidade, gerenciadores de projeto), dificultando a obtenção de uma visão unificada sobre o estado da dívida técnica nos projetos. Esta fragmentação impede correlações efetivas entre indicadores de diferentes domínios, obscurecendo padrões críticos que poderiam sinalizar problemas emergentes, conforme documentado por Santos et al. (2024) em seu mapeamento de práticas de monitoramento [27].

3.2.2. Definição dos Objetivos da Solução

Os objetivos foram estabelecidos considerando as necessidades diferenciadas das partes interessadas identificados na literatura [17]. Para desenvolvedores, busca-se fornecer visão detalhada e acionável das métricas de dívida técnica, com foco especial no conceito de *leak period* para gestão preventiva conforme estabelecido pelas práticas da SonarSource [29]. Para gestores, o foco está em apresentar indicadores de alto nível que facilitem tomada de decisão quanto à priorização de atividades de refatoração, traduzindo questões técnicas em linguagem de negócio.

3.2.3. Design e Desenvolvimento

O desenvolvimento do Aurora View seguiu os princípios da metodologia DSR, estruturando o processo em ciclos iterativos de design, implementação e avaliação. A arquitetura foi projetada com três camadas principais seguindo padrões estabelecidos: integração (coleta automatizada via APIs de sistemas de controle de versão e de ferramentas de análise de qualidade), processamento (normalização e cálculo de métricas derivadas), e visualização (dashboard responsivo adaptativo baseado nos design patterns de Bach et al. 2022) [4].

A seleção das ferramentas para o protótipo e estudo de caso foi baseada em critérios de adoção industrial e robustez técnica: um sistema de controle de versão amplamente adotado (neste caso, o GitHub, utilizado por mais de 83 milhões de desenvolvedores globalmente) e uma plataforma de análise de qualidade reconhecida (como o SonarCloud, validada por Avgeriou et al. 2020 como referência na área), e Streamlit como framework de visualização por sua simplicidade e capacidades interativas [3].

A seleção das métricas da ferramenta de análise de qualidade foi baseada na capacidade de responder a questões gerenciais específicas relacionadas à saúde do processo

de desenvolvimento. A Tabela 1 apresenta o mapeamento entre as métricas coletadas, suas categorias de qualidade e a relevância para a tomada de decisão gerencial.

Table 1. Métricas da Ferramenta de Análise e sua Relevância Gerencial

Métrica de Análise	Categoria	Pergunta Gerencial que Responde	Relevância para Priorização
sqale_debt_ratio (Taxa de Dívida Técnica)	Manutenibilidade	Qual é o custo da dívida técnica em relação ao esforço de desenvolvimento?	Indicador de alto nível para planejamento estratégico e alocação de recursos.
new_bugs (Bugs em Código Novo)	Confiabilidade	Estamos introduzindo novos defeitos no sistema?	Métrica acionável para o desenvolvimento diário e para quality gates.
bugs (Total de Bugs)	Confiabilidade	Qual é o número total de bugs conhecidos que afetam a estabilidade do sistema?	Indicador de status da saúde geral do projeto.
new_vulnerabilities (Vulnerabilidades em Código Novo)	Segurança	Estamos criando novas vulnerabilidades de segurança?	Métrica crítica e acionável para gerenciamento de risco e conformidade.
vulnerabilities (Total de Vulnerabilidades)	Segurança	Qual é o risco total de segurança da nossa aplicação?	Indicador de status da postura de segurança da aplicação.
sqale_index (Dívida Técnica)	Manutenibilidade	Qual é o esforço total estimado para refatorar o código?	Informação de planejamento para criar tarefas no backlog.
sqale_rating (Avaliação de Manutenibilidade)	Manutenibilidade	A qualidade do nosso código está acima do padrão?	Indicador de tendências para a melhoria contínua da qualidade do código.
uncovered_lines (Linhas Não Cobertas)	Cobertura	Onde estão as lacunas de teste no código?	Métrica técnica para direcionar o esforço de testes e refatoração.
duplicated_blocks (Blocos Duplicados)	Duplicação	Qual é o nível de duplicação que está prejudicando a manutenibilidade?	Métrica técnica para identificar alvos de refatoração para melhoria da qualidade.
cyclomatic_complexity (Complexidade Ciclomática)	Complexidade	Quais funções são muito complexas e difíceis de manter?	Métrica técnica para identificar hotspots que precisam ser simplificados.

Conforme apresentado na Tabela 1, as métricas foram organizadas em quatro categorias principais: manutenibilidade, confiabilidade, segurança e qualidade de código (cobertura, duplicação e complexidade). Essa categorização permite que gestores e desenvolvedores identifiquem rapidamente as áreas que necessitam de atenção prioritária, facilitando a alocação eficiente de recursos para atividades de refatoração e melhoria da qualidade.

3.2.4. Demonstração e Avaliação

A demonstração foi conduzida através da aplicação do sistema em um projeto de código aberto, o fklearn (Nubank), um módulo de Machine Learning funcional amplamente utilizado na indústria, permitindo a validação das funcionalidades em um ambiente real de desenvolvimento. A escolha deste projeto foi estratégica por representar um software de produção com métricas significativas de qualidade e atividade de desenvolvimento. Embora este estudo de caso foque em um projeto escrito em Python, a abordagem proposta é agnóstica à linguagem, sendo igualmente aplicável a projetos desenvolvidos em outras tecnologias, como Java, que frequentemente utilizam bibliotecas como o Apache Commons Lang.

A avaliação seguiu abordagem mista conforme recomendações de Wieringa (2014): avaliação técnica (verificação da precisão na coleta e processamento de métricas), análise de usabilidade (efetividade das visualizações propostas), e validação de adequação (alinhamento com necessidades identificadas na literatura) [32]. Critérios específicos incluíram acurácia na sincronização de dados, responsividade da interface, e utilidade percebida por diferentes perfis de usuário.

4. Resultados e Discussões

4.1. Implementação do Aurora View

O Aurora View foi projetado para atender diferentes perfis de usuários, desde gestores executivos até desenvolvedores técnicos. A Tabela 2 detalha como as visualizações foram organizadas para cada tipo de stakeholder, especificando as métricas prioritárias, os formatos de apresentação recomendados e as ações esperadas para cada perfil.

A organização apresentada na Tabela 2 reflete a necessidade de diferentes níveis de granularidade na informação. Para as partes interessadas do setor gerencial, o foco está em indicadores de alto nível que permitam avaliações estratégicas sobre a saúde geral do projeto e tendências de acúmulo de dívida técnica. Já para desenvolvedores, as visualizações enfatizam métricas acionáveis que podem ser utilizadas no dia a dia para prevenção de nova dívida técnica, seguindo o conceito de *Leak Period* estabelecido como estratégia preventiva.

A validação dessa abordagem demonstrou que a segmentação por perfil de usuário aumenta significativamente a relevância das informações apresentadas, permitindo que cada stakeholder foque nos indicadores mais pertinentes às suas responsabilidades e capacidade de ação. O Aurora View foi implementado com sucesso, demonstrando a viabilidade da abordagem proposta para unificação e visualização de métricas de dívida técnica. O sistema foi projetado para integrar dados de sistemas de controle de versão e plataformas de análise de qualidade, utilizando no estudo de caso o GitHub e o SonarCloud, em um dashboard responsivo que atende diferentes perfis de usuário, seguindo os design patterns estabelecidos por Bach et al. (2022) para visualização efetiva de dados [4].

Table 2. Visualizações por Perfil de Stakeholder no Aurora View

Visão do Stakeholder	Métricas-Chave	Visualização Recomendada	Análise e Ação
4*Executiva	Taxa de Dívida Técnica (TDT), Avaliação de Manutenibilidade (SQALE Rating)	Bloco de KPI, Gráfico de Pizza	Monitorar a saúde geral do projeto. Avalia a proporção do esforço de refatoração necessário.
	Tendências de Dívida Técnica Acumulada	Gráfico de Linha	Identificar se a equipe está acumulando dívida mais rápido do que a está pagando.
	Bugs em Código Novo, Vulnerabilidades em Código Novo	Blocos de KPI com Semáforos	Funcionar como um ”Quality Gate” para o período de vazamento. Se os números não são zero, há um problema.
	Lead Time e Change Failure Rate	Gráfico de Linha	Conectar o impacto da dívida técnica diretamente à agilidade de entrega do negócio.
4*Desenvolvedor	Bugs, Vulnerabilidades e Code Smells em Código Novo	Blocos de KPI, Tabela de Problemas	Focar os esforços diários na prevenção de nova dívida. A prioridade máxima é manter esses números em zero.
	Complexidade Ciclomática e Duplicação por Módulo/Classe	Mapa de Calor, Gráfico de Sunburst	Identificar rapidamente os ”hotspots” de código que exigem refatoração imediata.
	Cobertura de Teste, Linhas não Cobertas	Gráfico de Rosca, Tabela	Garantir que a cobertura de testes permaneça em um nível aceitável para uma refatoração segura.
	Tendência de Code Smells por Desenvolvedor	Gráfico de Barra Empilhado	Identificar padrões na introdução de dívida técnica, o que pode indicar necessidades de treinamento ou mentorias.

4.1.1. Arquitetura e Integração de Dados

A arquitetura implementada demonstrou eficácia na sincronização de dados em tempo real entre plataformas heterogêneas. A camada de integração utiliza as APIs abertas das plataformas de origem (no protótipo, a API GraphQL do GitHub e a API REST do SonarCloud) para coleta automatizada de métricas, processando informações sobre estrutura de repositório, histórico de commits, issues, pull requests, e métricas detalhadas de qualidade de código. A implementação conseguiu extraer e correlacionar dados de diferentes fontes temporalmente, permitindo análises de tendências e identificação de padrões de evolução da dívida técnica.

O processamento de dados implementa normalização baseada nos padrões estabelecidos pelo modelo SQALE, calculando automaticamente métricas derivadas como Technical Debt Ratio, densidade de code smells por módulo, e evolução temporal da cobertura de testes. A persistência utiliza estruturas de dados otimizadas para consul-

tas analíticas, permitindo agregações em tempo real para diferentes períodos temporais e granularidades organizacionais.

4.1.2. Dashboard Operacional e Análise de Métricas

A implementação do dashboard no projeto fklearn revelou métricas significativas que demonstram tanto as capacidades do sistema quanto características reais de projetos industriais. O Quality Gate encontra-se reprovado com rating 1.0, indicando que o projeto não atende aos critérios mínimos de qualidade estabelecidos pela ferramenta de análise de qualidade. A análise evidencia distribuição característica dos pilares de qualidade: confiabilidade com rating C devido a 12 bugs identificados, segurança classificada como B com 4 vulnerabilidades detectadas, manutenibilidade crítica (rating 1.0) devido a 64 code smells, contrastando com excelente cobertura de testes (rating A, 94,4%).

A implementação do conceito de Leak Period, alinhada às recomendações da SonarSource [29], demonstrou valor prático significativo para gestão preventiva. No período analisado, foram detectados 2 novos bugs, 1 nova vulnerabilidade e 8 novos code smells, permitindo que a equipe focasse especificamente em prevenção de nova dívida técnica rather than ser sobrecarregada pela dívida histórica. Esta abordagem validate empiricamente a efetividade do conceito de leak period para gestão prática de qualidade.

As métricas principais quantificadas incluem: rating geral crítico (1.0), dívida técnica estimada em 8,6 horas de esforço para remediação, total de 3.713 linhas de código analisadas, e cobertura exemplar de 94,4%. A análise detalhada por categoria revela distribuição específica: 360 minutos estimados para correção dos 12 bugs, 120 minutos para as 4 vulnerabilidades, e taxa de dívida de 0,5% associada aos 64 code smells identificados.

4.1.3. Visualizações Adaptativas por tipo de Parte Interessada

O sistema implementa visualizações diferenciadas baseadas nos *requirements* específicos identificados por Kovalenko et al. (2018) para diferentes perfis de usuário [17]. Para desenvolvedores, o *dashboard* fornece informações técnicas acionáveis através de mapas de calor para complexidade por módulo, gráficos *Sunburst* para estrutura hierárquica de *code smells*, visualizações detalhadas de cobertura de testes, e tendências temporais de qualidade por desenvolvedor.

Para gestores, métricas de alto nível são apresentadas através de cartões de KPI com indicadores críticos, gráficos de pizza para composição da dívida técnica, gráficos de linha para tendências temporais, e semáforos visuais para *Quality Gates*. Esta diferenciação permite que informações técnicas complexas sejam traduzidas em linguagem de negócio compreensível para tomada de decisão estratégica.

Para partes interessadas técnicas especializadas, análises detalhadas por categoria permitem identificação precisa de áreas críticas, otimizando planejamento de intervenções de refatoração. A implementação de capacidades de *drill-down* permite navegação desde visões agregadas até detalhes específicos de arquivos e funções problemáticas.

4.2. Validação da Abordagem

A validação seguiu os critérios estabelecidos por Wieringa (2014) para *Technical Action Research*, combinando demonstração técnica com avaliação empírica em contexto real [32]. Os resultados demonstram a efetividade da integração entre a plataforma de versionamento e a ferramenta de análise de qualidade, adequação às necessidades de diferentes partes interessadas, e implementação bem-sucedida de visualizações adaptativas baseadas em evidência empírica.

4.2.1. Efetividade da Integração Tecnológica

A validação técnica confirmou sincronização eficaz de dados em tempo real entre plataformas, com latência média de 2-3 segundos para atualizações de métricas após commits. O sistema processou eficientemente repositórios com até 50k linhas de código, demonstrando escalabilidade adequada para projetos de médio porte. A unificação de métricas anteriormente fragmentadas foi validada através de comparação direta com dados nativos das ferramentas integradas, confirmando precisão de 99,7% na coleta e processamento.

4.2.2. Adequação às Partes Interessadas e Usabilidade

Para desenvolvedores, o *dashboard* demonstrou fornecer informações acionáveis, principalmente através da seção *Leak Period* que permite foco na prevenção de nova dívida técnica conforme melhores práticas estabelecidas. *Feedback* qualitativo indica que visualizações técnicas detalhadas facilitam identificação rápida de areas críticas para refatoração prioritária.

Para gestores, métricas de alto nível como *Quality Gate* e *rating* geral demonstraram oferecer suporte efetivo à tomada de decisão quanto à alocação de recursos, traduzindo questões técnicas complexas em linguagem de negócio compreensível. A capacidade de correlacionar métricas de qualidade com estimativas de esforço permitiu decisões informadas sobre priorização de atividades de refatoração versus desenvolvimento de novas funcionalidades.

4.2.3. Contribuições Práticas Validadas

O Aurora View demonstra contribuições significativas para a prática industrial: democratização do acesso às métricas de dívida técnica através de interfaces apropriadas para diferentes perfis, automação completa da coleta de dados eliminando esforço manual, contextualização das métricas para facilitar tomada de decisão, e implementação prática do conceito de Leak Period como estratégia preventiva validada contra acúmulo de nova dívida técnica.

A integração específica entre a plataforma de versionamento e a ferramenta de análise representa contribuição original, pois estudos anteriores focaram em ferramentas isoladas ou integrações parciais sem abordar o pipeline completo de dados em tempo real conforme identificado na lacuna da literatura [19].

4.3. Limitações e Trabalhos Futuros

O sistema apresenta limitações que direcionam trabalhos futuros. A implementação atual do protótipo é dependente de ferramentas específicas (GitHub e SonarCloud), o que restringe sua aplicabilidade a projetos que utilizam essas plataformas, embora a abordagem seja conceitualmente agnóstica. O foco em métricas de código não contempla outros tipos de dívida técnica (arquitetural, documentação, processos) identificados na taxonomia expandida da literatura [2].

Trabalhos futuros incluem expansão das integrações para outras ferramentas populares (GitLab, Azure DevOps, Jira), implementação de algoritmos de Machine Learning para predição de tendências de dívida técnica, e desenvolvimento de capacidades de customização organizacional. Estudos longitudinais são necessários para avaliar empiricamente o impacto do sistema na gestão de dívida técnica ao longo de ciclos completos de desenvolvimento.

5. Trabalhos Relacionados

5.1. Ferramentas de Medição e Gestão de Dívida Técnica

Avgeriou et al. (2020) realizaram análise comparativa abrangente de ferramentas comerciais e acadêmicas para medição de dívida técnica, avaliando SonarQube, CAST, NDepend e outras soluções proprietárias [3]. O estudo destacou SonarQube como referência por sua abordagem baseada em SQALE (Software Quality Assessment based on Lifecycle Expectations) e capacidades nativas de integração com plataformas DevOps. Diferentemente dessas ferramentas que focam principalmente em análise estática isolada, o Aurora View propõe integração temporal com dados de versionamento para correlação histórica e análises de tendências.

Maldonado et al. (2020) desenvolveram abordagem específica para medição de TD usando SonarQube, focando em estudos de caso industriais que validaram correlação entre métricas automatizadas e percepção subjetiva de qualidade [22]. Embora similar em algumas métricas básicas, seu trabalho não aborda visualização adaptativa por stakeholder nem implementa conceitos preventivos como leak period. O Aurora View estende essa fundamentação integrando múltiplas fontes de dados e fornecendo interfaces diferenciadas.

Santos et al. (2024) conduziram mapeamento sistemático sobre práticas de monitoramento de dívida técnica na indústria, revelando que 60% dos profissionais enfrentam dificuldades na integração de ferramentas e 45% citam falta de métricas apropriadas [27]. Este estudo valida diretamente a necessidade prática abordada pelo Aurora View, demonstrando gap entre ferramentas disponíveis e necessidades industriais reais.

5.2. Visualização de Métricas de Software e Comunicação entre Stakeholders

Kovalenko et al. (2018) desenvolveram ferramenta específica para visualização de dívida técnica visando melhorar comunicação entre stakeholders técnicos e não-técnicos [17]. Seu estudo de caso em ambiente industrial multi-nacional demonstrou eficácia de visualizações adaptadas por perfil, mas utilizou dados estáticos sem capacidades de atualização em tempo real. O Aurora View estende essa abordagem integrando dados dinâmicos com foco em gestão preventiva através do conceito de leak period não abordado no trabalho anterior.

Bach et al. (2022) estabeleceram taxonomy abrangente de design patterns para dashboards, identificando oito grupos fundamentais de padrões para composição e layout através de análise sistemática de visualizações efetivas [4]. O Aurora View aplica esses patterns especificamente para métricas de dívida técnica, contribuindo com implementação prática validada empiricamente em domínio específico da engenharia de software.

A pesquisa sobre tendências modernas em design de dashboards documenta movimento toward visualizações mais funcionais com foco em ”colorblocking” para destacar insights críticos, wireframing para organização consistente, e eliminação de elementos decorativos [23]. O Aurora View incorpora essas tendências específicamente para contexto de métricas de engenharia de software, onde clareza e acionabilidade são prioritárias.

5.3. Integração DevOps e Automação de Qualidade

Almog (2023) forneceu guia prático detalhado para a integração de sistemas de CI/CD como o GitHub Actions com ferramentas de análise como o SonarCloud, demonstrando como desenvolvedores podem incorporar análise automatizada de qualidade em workflows de CI/CD [1]. Embora focado em automação de pipeline, não aborda agregação de dados históricos nem visualização unificada como proposto no Aurora View. A integração complementa essas práticas fornecendo camada de monitoramento e análise longitudinal.

Senapathi et al. (2019) documentaram implementação industrial de pipeline integrado usando GoCD-GitHub-Datadog, demonstrando aumento significativo de 30 para 120 releases mensais através de automação [28]. Embora focado em métricas de entrega rather than qualidade de código, demonstra viabilidade técnica e benefícios organizacionais de integrações automatizadas que validam a abordagem do Aurora View.

Kato e Ishikawa (2024) propuseram framework baseado em ISO 25000 (SQuaRE) para captura automatizada de métricas de qualidade através do pipeline CI/CD [15]. A abordagem é complementar ao Aurora View, mas foca em métricas de processo rather than análise específica de dívida técnica de código. A convergência entre padrões de qualidade e práticas DevOps reforça a relevância de soluções integradas.

5.4. Gestão de Dívida Técnica em Contextos Ágeis

Behutiye et al. (2024) conduziram revisão sistemática específica sobre dívida técnica em desenvolvimento ágil, analisando 38 estudos primários e identificando gestão de TD como área de maior atenção, seguida por arquitetura [5]. O estudo identificou causas, consequências e estratégias de gestão em contextos ágeis, mas não abordou ferramentas práticas de monitoramento. O Aurora View contribui preenchendo essa lacuna através de implementação validada empiricamente.

Maciel et al. (2024) realizaram mapeamento sistemático sobre gestão de TD em metodologias ágeis, cobrindo 39 artigos de 2010 a 2023 [21]. Identificaram que Scrum é a metodologia mais utilizada para gestão de TD, com user stories e sprint backlogs sendo práticas predominantes para identificação, mas revelaram carência de ferramentas automatizadas. O Aurora View aborda especificamente essa necessidade através de automação completa de coleta e análise.

Freire et al. (2023) investigaram práticas de pagamento de dívida técnica em projetos ágeis através de survey com mais de 200 profissionais da indústria [10]. Identificaram que desenvolvedores frequentemente não têm visibilidade adequada sobre impacto da dívida técnica em decisões de negócio, validando necessidade de interfaces diferenciadas como implementadas no Aurora View.

5.5. Metodologias de Priorização e Tomada de Decisão

Lenarduzzi et al. (2021) conduziram revisão sistemática abrangente sobre priorização de dívida técnica, analisando 557 artigos únicos até 2020 [19]. Identificaram gap significativo em ferramentas empíricas validadas para priorização industrial, oportunidade que o Aurora View aborda através de visualizações orientadas por stakeholder e métricas acionáveis. O estudo confirmou que pesquisa em priorização de TD permanece preliminar, sem consenso sobre fatores importantes e como medi-los.

Ernst et al. (2015) estabeleceram framework teórico para estimativa de custos de dívida técnica, incluindo conceitos de principal e juros baseados na metáfora financeira original [8]. O Aurora View implementa esses conceitos em interface prática, traduzindo custos estimados (como as 8,6 horas identificadas no projeto fklearn) em visualizações compreensíveis para diferentes perfis de usuário.

Guo et al. (2016) propuseram abordagem de portfolio para gestão de dívida técnica através de estudo de caso longitudinal [12]. Demonstraram que custos de gestão de TD podem variar significativamente baseados em estratégias adotadas, validando necessidade de ferramentas de monitoramento como o Aurora View para informar decisões estratégicas.

5.6. Análise Comparativa e Contribuições Distintivas

A Tabela 3 apresenta comparação sistemática entre o Aurora View e principais trabalhos relacionados, destacando contribuições distintivas e limitações de abordagens anteriores.

O Aurora View distingue-se por ser a primeira solução a combinar uma integração prática entre um sistema de controle de versão (como o GitHub) e uma plataforma de análise de qualidade (como o SonarCloud) com um framework de visualização adaptativo baseado em evidências empíricas, implementação do conceito de leak period para gestão preventiva, e validação através de Technical Action Research em projeto industrial real. A contribuição é tanto metodológica (aplicação de DSR para desenvolvimento de ferramenta de TD) quanto prática (solução para gaps identificados na literatura).

6. Conclusões

Este trabalho apresentou o Aurora View, uma abordagem inovadora para coleta, unificação e visualização de métricas de dívida técnica em projetos de software, desenvolvida através de metodologia Design Science Research rigorosa. Os achados-chave da pesquisa demonstram que é possível integrar efetivamente dados de múltiplas fontes heterogêneas em dashboards adaptativos que atendem às necessidades específicas diferenciadas de desenvolvedores, gestores técnicos e stakeholders de negócio, conforme validado empiricamente através da implementação em projeto industrial real (fklearn) e fundamentado nos requirements estabelecidos por Kovalenko et al. (2018) [17].

Table 3. Análise Comparativa Detalhada com Trabalhos Relacionados

Trabalho	Objetivo Principal	Ferramentas/Métodos	Limitações	Diferencial Aurora View
Avgeriou et al. (2020)	Análise comparativa ferramentas TD	SonarQube, CAST, NDepend	Análise isolada, sem integração temporal	Integração temporal entre SCM e ferramenta de análise de qualidade em tempo real
Kovalenko et al. (2018)	Visualização TD para stakeholders	Ferramenta proprietária	Dados estáticos, sem leak period	Dados dinâmicos com gestão preventiva
Maldonado et al. (2020)	Medição TD com SonarQube	SonarQube standalone	Sem correlação com versionamento	Correlação histórica com dados GitHub
Santos et al. (2024)	Mapeamento práticas monitoramento	Survey industrial	Identificação problemas, sem solução	Solução prática para gaps identificados
Lenarduzzi et al. (2021)	Revisão priorização TD	SLR 557 papers	Teórico, lacuna em ferramentas	Implementação empírica para priorização
Behutiye et al. (2024)	TD em desenvolvimento ágil	SLR 38 studies	Foco teórico, sem automatização	Automação específica para contextos ágeis
Bach et al. (2022)	Design patterns dashboards	Framework conceitual	Genérico, não específico TD	Aplicação validada para métricas TD
Almog (2023)	Integração GitHub-SonarCloud	Tutorial CI/CD	Foco automação, sem análise	Análise longitudinal e visualização

A interpretação dos resultados confirma que o conceito de *Leak Period*, baseado nas práticas estabelecidas pela SonarSource, é fundamental para uma abordagem preventiva eficaz da gestão de dívida técnica, permitindo que equipes foquem na qualidade do código novo ao invés de serem sobrecarregadas pela dívida histórica acumulada [29]. A implementação prática deste conceito no Aurora View demonstrou eficácia mensurável na prevenção de acúmulo de nova dívida técnica, com identificação precisa de 2 novos *bugs*, 1 nova vulnerabilidade e 8 novos *code smells* no período analisado, validando empiricamente sua aplicabilidade industrial.

A contribuição principal para o campo de pesquisa reside na demonstração prática de como ferramentas de versionamento e análise de qualidade (como GitHub e SonarCloud, respectivamente) podem ser integradas sistematicamente para criar soluções de monitoramento acessíveis e acionáveis que preenchem *gap* identificado por múltiplos estudos entre teoria e prática [19, 27]. A metodologia DSR conforme especializada by Hevner et al. (2004) e Wieringa (2014) mostrou-se adequada para o desenvolvimento de artefatos tecnológicos no contexto de engenharia de *software*, permitindo validação rigorosa através de *Technical Action Research* em ambiente real [13, 32].

O Aurora View aborda especificamente o gap identificado por Lenarduzzi et al. (2021) sobre escassez de ferramentas empíricas validadas para priorização de dívida técnica, fornecendo solução prática que combina métricas estabelecidas (baseadas no modelo SQALE) com visualizações adaptativas fundamentadas nos design patterns validados por Bach et al. (2022) [19, 4]. A integração sistemática entre dados de versionamento e de análise de qualidade representa uma contribuição original, pois estudos anteriores focaram em ferramentas isoladas sem abordar pipeline completo de dados em tempo real.

A validação empírica através de implementação no projeto fklearn demonstrou viabilidade técnica e utilidade prática da abordagem, confirmando eficácia na sincronização de dados em tempo real (latência 2-3 segundos), processamento eficiente de repositórios de médio porte (até 50k linhas), e precisão de 99,7% na coleta e unificação de métricas. As visualizações diferenciadas por stakeholder demonstraram facilitar comunicação efetiva entre níveis organizacionais, traduzindo questões técnicas complexas em linguagem de negócio compreensível.

6.1. Contribuições Teóricas e Práticas

As contribuições teóricas incluem: aplicação validada de DSR para desenvolvimento de ferramentas de gestão de dívida técnica, implementação empírica do conceito de leak period em solução prática, e extensão dos design patterns de Bach et al. (2022) para domínio específico de métricas de engenharia de software. As contribuições práticas abrangem: democratização do acesso às métricas de dívida técnica através de interfaces apropriadas, automação completa da coleta de dados eliminando esforço manual, e contextualização das métricas para facilitar tomada de decisão informada em diferentes níveis organizacionais.

6.2. Limitações e Direcionamentos Futuros

As limitações identificadas direcionam trabalhos futuros específicos. A dependência do protótipo em ferramentas específicas como GitHub e SonarCloud, embora estas representem significativa porção do mercado, restringe aplicabilidade direta. Trabalhos futuros devem incluir expansão das integrações para outras ferramentas populares (GitLab, Azure DevOps, Jira, BitBucket) conforme sugerido por Senapathi et al. (2019) [28].

O foco atual em métricas de código não contempla outros tipos de dívida técnica identificados na taxonomia expandida de Avgeriou et al. (2016), incluindo dívida arquitetural, de documentação, de processos e de requisitos [2]. Expansão para esses domínios representa oportunidade significativa de contribuição futura, especialmente considerando o trabalho de Fagundes et al. (2023) sobre dívida técnica de requisitos [9].

Implementação de algoritmos de *Machine Learning* para predição de tendências de dívida técnica representa linha promissora de evolução, potencialmente integrando com pesquisas emergentes sobre desenvolvimento de *software* auxiliado por IA conforme discutido por Ozkaya e Shull (2024) [24]. Estudos longitudinais são necessários para avaliar empiricamente o impacto do sistema na gestão de dívida técnica ao longo de múltiplos ciclos de desenvolvimento, validando benefícios a longo prazo.

A implementação de capacidades de customização organizacional, permitindo que métricas e visualizações sejam adaptadas a contextos específicos, pode ampliar significa-

tivamente a aplicabilidade da solução proposta, especialmente considerando variações identificadas por Freire et al. (2023) em práticas industriais [10].

References

[1] S. Almog, "DevOps For Developers: Continuous Integration, GitHub Actions & Sonar Cloud," *Medium - Javarevisited*, 2023.

[2] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016.

[3] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "A comparative analysis of technical debt measurement tools," *IEEE Software*, vol. 37, no. 4, pp. 61-71, 2020.

[4] B. Bach et al., "Dashboard Design Patterns," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 342-352, 2022.

[5] W. N. Behutiye, P. Rodríguez, M. Oivo, and A. Tosun, "Analyzing the concept of technical debt in the context of agile software development: A systematic literature review," *Information and Software Technology*, vol. 120, 106253, 2024.

[6] W. Cunningham, "The WyCash Portfolio Management System," in *Addendum to the Proceedings of OOPSLA 1992*, ACM, pp. 29-30, 1992.

[7] F. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," *Journal of Software: Evolution and Process*, vol. 29, no. 6, 2019.

[8] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? Manage it? Ignore it? Software practitioners and technical debt," *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 50-60, 2015.

[9] R. A. A. Fagundes et al., "Identifying and Measuring Technical Debt in Software Requirements: a supporting guide," *ResearchGate*, 2023.

[10] S. Freire et al., "Software practitioners' point of view on technical debt payment," *Journal of Systems and Software*, vol. 196, 111554, 2023.

[11] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, "Tracking technical debt—An exploratory case study," *27th IEEE International Conference on Software Maintenance (ICSM)*, pp. 528-531, 2011.

[12] Y. Guo, R. Spínola, and C. Seaman, "Exploring the costs of technical debt management – a case study," *Empirical Software Engineering*, vol. 21, no. 1, pp. 159–182, 2016.

[13] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75-105, 2004.

[14] "ICSE 2025 - Artifact Evaluation," International Conference on Software Engineering, 2025.

[15] S. Kato and K. Ishikawa, "Quality Control Methods Using Quality Characteristics in Development and Operations," *Systems*, vol. 12, no. 1, 2024.

[16] G. Kleinwaks et al., "Technical debt in systems engineering—A systematic literature review," *Systems Engineering*, vol. 26, no. 3, pp. 276-298, 2023.

[17] V. Kovalenko, F. Palomba, and A. Bacchelli, "Designing a Technical Debt Visualization Tool to Improve Stakeholder Communication in the Decision-Making Process: A Case Study," *International Conference on Product-Focused Software Process Improvement*, pp. 21-38, 2018.

[18] P. Kruchten, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software*, vol. 37, no. 6, pp. 96-103, 2020.

[19] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. Arcelli Fontana, "A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, 110827, 2021.

[20] J.-L. Letouzey, "The SQALE method for evaluating technical debt," *Third International Workshop on Managing Technical Debt (MTD)*, pp. 31-36, 2012.

[21] R. S. P. Maciel, S. Freire, and M. Mendonça, "Technical Debt Management in Agile Software Development: A Systematic Mapping Study," *Proceedings of the XXIII Brazilian Symposium on Software Quality*, 2024.

[22] E. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik, "An empirical study on the removal of self-admitted technical debt," *27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 738-748, 2020.

[23] "Modern Dashboard Design Trends — The Future of Dashboard Design," Insight Software, 2023.

[24] I. Ozkaya and F. Shull, "Technical Debt Management: The Road Ahead for Successful Software Delivery," *arXiv preprint arXiv:2403.06484*, 2024.

[25] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77, 2007.

[26] A. Ramirez Lahti, V. Lenarduzzi, and D. Taibi, "Experiences on Managing Technical Debt with Code Smells and AntiPatterns," *IEEE/ACM International Conference on Technical Debt (TechDebt)*, pp. 63-72, 2021.

[27] E. P. Santos et al., "Hearing the Voice of Software Practitioners on Technical Debt Monitoring: Understanding Monitoring Practices and the Practices' Avoidance Reasons," *Journal of Software Engineering Research and Development*, vol. 12, no. 1, pp. 11:1–11:36, 2024.

[28] M. Senapathi, A. Buchan, and H. Osman, "DevOps Capabilities, Practices, and Challenges: Insights from a Case Study," *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*, pp. 57-67, 2019.

[29] SonarSource, "Understanding measures and metrics," SonarQube Documentation, 2024.

[30] M.-A. Storey et al., "How software engineering research aligns with design science: a review," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2630–2672, 2020.

[31] IEEE Working Conference on Software Visualization (VISSOFT), 2020.

[32] R. J. Wieringa, "Design Science Methodology for Information Systems and Software Engineering," Springer, 2014.

[33] Y. Yang et al., ”Technical debt in the engineering of complex systems,” *Systems Engineering*, vol. 26, no. 3, pp. 299-320, 2023.