UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE

BugAwareRetro: Ferramenta de apoio às reuniões de Retrospectiva

RITA DE CÁSSIA LINO LOPES

NATAL/RN, BRASIL

2025

RITA DE CÁSSIA LINO LOPES

BugAwareRetro: Ferramenta de apoio às reuniões de Retrospectiva

Monografia apresentada ao Curso de Bacharelado em Engenharia de Software do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientadora: Profa. Dra. Roberta de Souza Coelho

Natal-RN
2025

Universidade Federal do Rio Grande do Norte - UFRN

Sistema de Bibliotecas - SISBI

Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Lopes, Rita de Cássia Lino.

BugAwareRetro: ferramenta de apoio às reuniões de retrospectiva / Rita de Cássia Lino Lopes. - 2025.

95 f.: il.

Monografia (graduação) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Curso de Bacharelado em Engenharia de Software. Natal, RN, 2025.

Orientação: Profa. Dra. Roberta de Souza Coelho.

1. Scrum - Monografia. 2. Retrospectiva - Monografia. 3. Qualidade de software - Monografia. 4. Melhoria contínua - Monografia. 5. Engenharia de software - Monografia. I. Coelho, Roberta de Souza. II. Título.

RN/UF/CCET CDU 004.414.32

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

Rita de Cássia Lino Lopes

BugAwareRetro: Ferramenta de apoio às reuniões de Retrospectiva

Monografia apresentada ao Curso de Bacharelado em Engenharia de Software do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho aprovado. Natal-RN, Brasil, 11 de julho de 2025:

_____

Profa. Dra. Roberta de Souza Coelho
Orientador

_____

Prof. Dr. Fernando Marques Figueira Filho
Examinador

_____

Profa. Dra. Márcia Jacyntha Nunes Rodrigues Lucena
Examinador

Natal-RN
2025

Dedico este trabalho ao Deus da minha vida,
Ao Senhor da minha Salvação.

AGRADECIMENTOS

Ao Deus da minha vida, Jesus. Àquele que nunca deixou de me ver. Àquele que me quis quando ninguém mais me quis. Àquele que me guiou e, outras vezes, me carregou pelos dias e noites desta história. Em tanto tempo de caminhada ao seu lado, me lembrei do começo, de quando Ele me chamou — nada seria o que é sem Ele. Só posso sorrir porque Ele viu o meu choro e se agradou de ter misericórdia de alguém como eu. Jesus é a força quando tenho medo, é a esperança em momentos vazios, é a luz em dias sombrios, é o ar quando não consigo respirar, é a água que me traz alívio, é o Caminho, a Verdade e a Vida. A Ele toda honra, toda glória e todo louvor. A Ele dedico tudo na completa certeza de que só foi possível porque Ele não desistiu de mim, Ele quis, levantou pessoas, me fortaleceu, trouxe direção e fez. Então, se lhe entreguei minhas derrotas, minhas vitórias são dele também.

À minha mãe. Mãezinha, sua vida e resiliência me inspiram a não desistir mesmo diante da falta de força e improbabilidades. Eu sei que, mesmo distante fisicamente, não deixou de sentir por mim, de orar, de cuidar e de se preocupar comigo.

Aos meus pastores. Suas lições, direções, exemplos, exortações e, especialmente, orações foram essenciais não apenas para que este trabalho fosse possível, mas também para que eu não desistisse tantas vezes ao longo do caminho. Obrigada por zelarem pela minha vida, por cuidarem de mim tanto quando estou de pé quanto quando estou quebrada. Sei que, se os senhores não tivessem entrado nesse desafio comigo, eu não teria conseguido. Os senhores me motivaram, me orientaram, alertaram, disciplinaram e amaram. As suas vidas são uma inspiração para mim!

Aos irmãos da IRC. Além de intercederem, me apoiaram durante a produção desta monografia. Vocês têm sido como ferro me afiando e me ensinando. Obrigada pelas orações e pelo serviço.

À minha professora orientadora Roberta de Souza Coelho. Além de orientações acadêmicas e excelência, a sua gentileza e paciência me motivaram a escrever toda esta monografia. Sua forma de compartilhar conhecimentos me ajuda a enxergar a docência como uma forma de inspirar sonhos. A senhora é uma inspiração para mim como pessoa e como docente. Obrigada!

À Universidade Federal do Rio Grande do Norte, instituição que faz parte da minha vida há alguns anos e na qual pude obter grandes aprendizados e conquistas.

RESUMO

A identificação e correção de defeitos representa uma parcela significativa do esforço total de desenvolvimento de software, podendo ser até 100 vezes mais cara após a entrega. Embora as retrospectivas sejam reconhecidas como mecanismo fundamental de melhoria contínua no Scrum, observa-se significativa divergência entre seu potencial teórico e implementação prática. Estudos empíricos revelam que a maioria das retrospectivas não aproveitam dados de projetos de software, fundamentando-se unicamente em percepções subjetivas. Esta abordagem compromete a qualidade das análises realizadas e a efetividade das ações de melhoria propostas. A ausência de rastreabilidade adequada entre defeitos, seus contextos de ocorrência e as reflexões conduzidas durante as retrospectivas constitui lacuna crítica que impede as equipes de transformarem experiências práticas em aprendizado sistematizado. Este trabalho tem como objetivo desenvolver uma ferramenta Web denominada Sistema BugAwareRetro, que proporcione suporte estruturado às reuniões de retrospectiva através da apresentação organizada de dados de defeitos. O sistema visa aprimorar a qualidade das retrospectivas e promover a melhoria contínua através da correlação entre dados quantitativos de falhas e percepções qualitativas das equipes. A metodologia adotada empregou o backend em Java-Spring Boot e frontend em React-TypeScript. O BugAwareRetro diferencia-se de ferramentas existentes como Jira, Azure DevOps e outras focadas em retrospectiva ao propor solução integrada que unifica rastreamento de bugs e gestão de retrospectivas, atuando durante a Sprint para registro de defeitos e na Cerimônia de Retrospectiva para apresentação organizada dos dados, proporcionando ciclo contínuo baseado em evidências empíricas concretas.

Palavras-chave: Retrospectiva. Melhoria contínua. Qualidade de Software. Scrum. Engenharia de Software.

ABSTRACT

The identification and correction of defects represent a significant portion of the total effort in software development, potentially costing up to 100 times more after deployment. Although retrospectives are recognized as a fundamental mechanism for continuous improvement in Scrum, there is a significant gap between their theoretical potential and practical implementation. Empirical studies reveal that most retrospectives do not leverage project data, relying solely on subjective perceptions. This approach undermines the quality of the analyses conducted and the effectiveness of the improvement actions proposed. The lack of adequate traceability between defects, their contexts of occurrence, and the reflections conducted during retrospectives constitutes a critical gap that prevents teams from transforming practical experiences into systematized learning. This study aims to develop a web-based tool called BugAwareRetro, which provides structured support for retrospective meetings through the organized presentation of defect data. The system seeks to enhance the quality of retrospectives and foster continuous improvement by correlating quantitative failure data with the team's qualitative insights. The methodology employed includes a Java Spring Boot backend and a React TypeScript frontend. BugAwareRetro differentiates itself from existing tools such as Jira, Azure DevOps, and other retrospective-focused platforms by proposing an integrated solution that unifies bug tracking and retrospective management. It operates throughout the Sprint for defect registration and during the Retrospective Ceremony to present the data in an organized manner, enabling a continuous cycle based on concrete empirical evidence.

Keywords: Retrospective. Continuous Improvement. Software Quality. Scrum. Software Engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E SIGLAS

SIGLA	SIGNIFICADO
ACID	Atomicity, Consistency, Isolation, Durability
ALM	Application Lifecycle Management
API	Application Programming Interface
BDD	Behavior-Driven Development
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DTO	Data Transfer Object
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
OAuth2	Open Authorization 2.0
PDCA	Plan-Do-Check-Act
QA	Quality Assurance
RCA	Root Cause Analysis
REST	Representational State Transfer
SSO	Single Sign-On
TDD	Test-Driven Development
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
Web	World Wide Web
WIP	Work In Progress
XSS	Cross-Site Scripting

SUMÁRIO

1 – INTRODUÇÃO

1.1 - CONTEXTUALIZAÇÃO

Desde a fase de concepção do software, fala-se enfaticamente sobre requisitos e agilidade. No entanto, pouca atenção é dada para os problemas introduzidos durante a codificação e identificados durante os testes de software. Segundo estudos da engenharia de software, a identificação e correção de defeitos representa uma parcela significativa do esforço total de desenvolvimento (BOEHM, 1981; NASA, 2010). Além disso, estudos clássicos demonstram que corrigir um defeito após a entrega pode ser até 100 vezes mais caro do que corrigi-lo durante as fases iniciais de requisitos e design (BOEHM, 1981).

Nos últimos anos, a adoção de metodologias ágeis tem crescido significativamente no desenvolvimento de software, sendo o Scrum uma das abordagens mais utilizadas por equipes de tecnologia. Sobre isso, dados do 17º State of Agile Report indicam que 71% dos respondentes utilizam práticas ágeis em seu ciclo de desenvolvimento de software (NOTTA, 2024), com o Scrum sendo uma das metodologias mais populares (ALMALKI, 2025). Com ciclos curtos e entregas frequentes, o Scrum promove maior adaptação às mudanças e foco contínuo na entrega de valor ao cliente.

Um dos pilares do Scrum é a cerimônia de retrospectiva da sprint, um momento dedicado à reflexão sobre o que funcionou bem, o que poderia ser melhorado e o que será ajustado para a próxima sprint. Conforme definido no Scrum Guide, a retrospectiva é uma oportunidade para a equipe Scrum inspecionar a si mesma e criar um plano de melhorias a serem aplicadas na próxima sprint (SCHWABER; SUTHERLAND, 2013, p. 12; SCHWABER; SUTHERLAND, 2020). No entanto, mesmo com a sua importância teórica, muitas equipes realizam essa atividade de forma superficial, sem registros estruturados ou análise de dados concretos que sustentem decisões de melhoria. Estudos empíricos sobre retrospectivas ágeis identificam desafios relacionados ao

comprometimento das equipes em implementar itens de ação definidos durante as retrospectivas.

Um aspecto crítico frequentemente negligenciado é a correlação entre defeitos identificados e a efetividade dos processos de desenvolvimento. Estudos empíricos sobre rastreabilidade demonstram que a rastreabilidade entre diferentes tipos de artefatos de software é fundamental para melhorar a correção e manutenibilidade do software (CHARALAMPIDOU et al., 2021). A falta de rastreabilidade adequada entre bugs, suas origens e impactos nas sprints compromete a capacidade de melhoria contínua das equipes.

Em um cenário de constante evolução tecnológica e alta competitividade, é fundamental que as equipes não apenas entreguem software, mas também aprendam continuamente com suas experiências para que a qualidade das entregas seja cada vez maior. Segundo o conceito de Learning Organization de Peter Senge, organizações que aprendem sistematicamente com suas experiências apresentam vantagem competitiva sustentável (SENGE, 1990; MERIDIAN UNIVERSITY, 2024; INFED, 2024). Nesse contexto, a utilização de ferramentas que organizem, consolidem e analisem dados das sprints pode transformar a retrospectiva em uma prática mais objetiva, estratégica e orientada à melhoria contínua.

Pesquisas recentes propõem a incorporação de sistemas de apoio à decisão em aplicações de retrospectivas ágeis, com o objetivo de fornecer soluções mais efetivas através da automatização e refinamento da aplicação prática de processos Ágeis/Scrum (ALMALKI, 2025; MILANI, 2025).

Diante disso, este trabalho propõe o desenvolvimento de uma ferramenta Web, chamada de BugAwareRetro, voltada para apoiar a realização de retrospectivas ágeis com base em informações estruturadas, registros de falhas e geração de relatórios analíticos. A proposta alinha-se com os princípios do Evidence-Based Software Engineering (CARTAXO et al, 2020), que preconiza o uso de evidências empíricas para fundamentar decisões de melhoria de processos.

O BugAwareRetro caracteriza-se como sistema de informação fundamentado nos princípios do Evidence-Based Software Engineering por três aspectos fundamentais: primeiro, sua concepção baseia-se em evidências empíricas (experiências) coletadas através de conversas aproximadamente vinte profissionais da indústria e academia, e análise sistemática da literatura especializada; segundo, promove a substituição de decisões baseadas exclusivamente em percepções subjetivas por análises que integram dados quantitativos de defeitos com insights qualitativos da equipe, abordando a lacuna identificada empiricamente onde aproximadamente 86% das retrospectivas não aproveitam dados de projeto de software (MATTHIES; DOBRIGKEIT, 2021); terceiro, fundamenta-se em pesquisas sobre eficácia de retrospectivas orientadas por dados, proporcionando correlação sistemática entre informações objetivas e percepções da equipe durante cerimônias ágeis, preenchendo lacuna crítica identificada na literatura especializada (MILANI et al., 2025).

1.2 - PROBLEMÁTICA E JUSTIFICATIVA

Embora a importância da retrospectiva no Scrum seja amplamente reconhecida como mecanismo fundamental de melhoria contínua, observa-se na prática uma significativa divergência entre o potencial teórico desta cerimônia e sua implementação efetiva nas organizações. Esta lacuna fica evidente particularmente na realidade cotidiana das equipes de desenvolvimento, na qual o valor transformador das retrospectivas frequentemente não se materializa.

Estudos empíricos revelam que a maioria das equipes ágeis conduz retrospectivas de maneira superficial, caracterizadas pela ausência de dados estruturados e pela predominância de discussões baseadas exclusivamente em percepções subjetivas dos participantes. Conforme demonstrado por Matthies e Dobrigkeit (2021), as retrospectivas frequentemente não incorporam dados concretos de projetos de software durante suas discussões, fundamentando-se unicamente nas impressões pessoais dos membros da equipe.

Essa abordagem compromete significativamente a qualidade das análises realizadas e a efetividade das ações de melhoria propostas, uma vez que decisões estratégicas são tomadas sem o devido embasamento empírico. Como consequência, as equipes enfrentam dificuldades substanciais para estabelecer correlações objetivas entre defeitos identificados, suas origens e os impactos resultantes nos processos de desenvolvimento, comprometendo tanto a capacidade de aprendizado organizacional quanto a melhoria contínua.

A ausência de rastreabilidade adequada entre defeitos, seus contextos de ocorrência e as reflexões conduzidas durante as retrospectivas constitui uma lacuna crítica que impede as equipes de transformarem experiências práticas em aprendizado sistematizado. Consequentemente, as retrospectivas resultam em discussões genéricas que não produzem insights acionáveis para o aprimoramento dos processos de desenvolvimento.

Diante deste cenário, evidencia-se a necessidade urgente de ferramentas especializadas que integrem dados quantitativos de defeitos com percepções qualitativas das equipes. Tais ferramentas proporcionariam uma base sólida para a condução de retrospectivas mais efetivas e orientadas à melhoria contínua baseada em evidências concretas, além de facilitar a integração de dados provenientes de múltiplas plataformas.

1.3 - OBJETIVOS

Esta monografia apresenta o seguinte objetivo geral: desenvolver uma ferramenta Web denominada Sistema BugAwareRetro, que proporcione suporte estruturado às reuniões de retrospectiva em projetos de desenvolvimento de software através da apresentação organizada de dados de defeitos. O sistema visa aprimorar a qualidade das retrospectivas e promover a melhoria contínua dos processos através da correlação entre dados quantitativos de falhas e percepções qualitativas das equipes durante o ciclo de vida das sprints.

Os objetivos específicos deste trabalho são:

- Desenvolver interface analítica que apresente métricas de defeitos através de dashboards e visualizações gráficas, incluindo taxa de defeitos, distribuição por severidade e análises comparativas por sprint.
- Estruturar a apresentação de dados quantitativos de defeitos durante cerimônias de retrospectiva, permitindo que equipes fundamentem discussões qualitativas em evidências concretas sobre falhas ocorridas durante as sprints e estabeleçam ações de melhoria contínua baseadas em análise dos dados e padrões identificados.
- Implementar sistema de visualização do ciclo de vida dos defeitos com capacidades de integração a ferramentas externas consolidadas como Azure Devops, garantindo rastreabilidade adequada e evitando duplicação de esforços nas organizações.

1.4 - CONTRIBUIÇÕES

Este trabalho propõe uma solução ao integrar elementos de análise de dados ao processo de retrospectiva ágil, muitas vezes tratado de forma subjetiva ou informal. A principal contribuição está na criação de uma ferramenta que organiza informações, identifica padrões e transforma experiências em aprendizado coletivo, gerando valor para a equipe de desenvolvimento.

Do ponto de vista acadêmico, o projeto contribui com o estudo da aplicação prática de conceitos da Engenharia de Software aliados ao Scrum e à melhoria contínua, fortalecendo o uso da tecnologia como apoio à gestão de times ágeis.

1.5 - ORGANIZAÇÃO DO DOCUMENTO

Este trabalho está estruturado em cinco capítulos, conforme descrito a seguir.

O Capítulo 1 – Introdução apresenta o tema da pesquisa, abordando a contextualização do problema, os objetivos do trabalho, as contribuições esperadas e a organização do documento.

Em seguida, o Capítulo 2 – Fundamentação Teórica reúne o embasamento conceitual necessário à proposta.

O Capítulo 3 – BugAwareRetro: Ferramenta de Apoio à Gestão de Bugs nas Reuniões de Retrospectiva descreve o sistema proposto

O Capítulo 4 – Desenvolvimento do Sistema apresenta os aspectos técnicos do desenvolvimento da aplicação

O Capítulo 5 – Considerações Finais apresenta os resultados alcançados, as dificuldades enfrentadas e as contribuições pessoais e profissionais decorrentes do trabalho.

2 – FUNDAMENTAÇÃO TEÓRICA

2.1 - METODOLOGIAS ÁGEIS E O SCRUM

As metodologias ágeis enfatizam ciclos iterativos de trabalho, feedback contínuo e flexibilidade para se adaptar a requisitos em mudança (MILANI et al., 2025). Uma característica fundamental das abordagens ágeis é a busca pela melhoria contínua do processo de desenvolvimento. Os métodos ágeis de desenvolvimento de software enfatizam a importância de usar retrospectivas continuamente de acordo com os princípios ágeis: "Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, então ajusta e adapta seu comportamento adequadamente" (LEHTINEN et al., 2017). Ou seja, a adaptação constante é intrínseca ao ágil, fomentando uma cultura de aprendizado e evolução a cada iteração do produto.

Dentre as diversas metodologias ágeis propostas, o Scrum tornou-se a mais amplamente difundida na indústria. Segundo o 17º State of Agile Report, baseado na pesquisa conduzida em 2023, 71% dos respondentes usam Ágil em seu ciclo de vida de desenvolvimento de software, enquanto a metodologia Ágil mais popular continua sendo o Scrum (SPICHKOVA et al., 2025).

Scrum é um framework que ajuda pessoas, equipes e organizações a gerar valor através de soluções adaptativas para problemas complexos (SCHWABER & SUTHERLAND, 2020). O Scrum organiza o desenvolvimento em ciclos curtos e fixos denominados sprints, que são eventos de duração fixa de um mês ou menos para criar consistência (SCHWABER & SUTHERLAND, 2020). Cada sprint resulta em um incremento entregável e completo na funcionalidade proposta do produto, promovendo entregas frequentes e feedback rápido.

O Scrum define papéis bem estabelecidos. A Equipe Scrum consiste em um Scrum Master, um Product Owner e Developers (SCHWABER & SUTHERLAND, 2020). Além disso, também encontramos confirmação desses papéis nos estudos empíricos: "A organização de desenvolvimento inclui desenvolvedores de software, desenvolvedores líderes, scrum masters e product owners" (LEHTINEN et al., 2017, p. 2415).

O framework define um conjunto de eventos (cerimônias) que se repetem a cada sprint: Sprint Planning, Daily Scrum, Sprint Review e Sprint Retrospective (SCHWABER & SUTHERLAND, 2020). Evidências empíricas confirmam o uso desses eventos: "Eles também conduzem daily stand-ups, demonstrações de sprint e retrospectivas" (LEHTINEN et al., 2017, p. 2416).

Essa cadência estruturada sustenta um modelo de processo empírico, baseado nos pilares da transparência, inspeção e adaptação contínua. O Scrum é fundado no empirismo, ou seja, na experiência e pensamento enxuto. O empirismo afirma que o conhecimento vem da experiência e da tomada de decisões baseadas no que é observado (SCHWABER & SUTHERLAND, 2020).

Em particular, a Retrospectiva de Sprint é o evento dedicado explicitamente à inspeção do processo de trabalho e adaptação do comportamento da equipe, sendo considerada por muitos autores e praticantes como um momento crítico para o sucesso contínuo de equipes ágeis. Reuniões de retrospectiva eficazes são vitais para garantir processos de desenvolvimento produtivos, pois fornecem os meios para as equipes de desenvolvimento de software Ágil discutirem e decidirem sobre melhorias futuras de sua colaboração (MATTHIES & DOBRIGKEIT, 2021).

De fato, pesquisas de campo indicam que a grande maioria das equipes realiza uma retrospectiva ao final de cada sprint. Uma vasta maioria dos respondentes (81%) a uma pesquisa de 2018 declarou que suas equipes realizavam uma reunião de Retrospectiva ao final de cada Sprint (MATTHIES & DOBRIGKEIT, 2021), reforçando que essa prática de reflexão estruturada está solidamente enraizada no dia a dia do desenvolvimento ágil moderno.

No Scrum, cada sprint inicia-se com a Sprint Planning (planejamento das funcionalidades e tarefas a realizar) e termina com duas cerimônias complementares: a Review, focada nos resultados do produto (demonstração do incremento desenvolvido e coleta de feedback do cliente), e a Retrospectiva, focada no processo e na dinâmica da equipe. É nessa última que o time Scrum inspeciona como foi o último sprint no que diz respeito a pessoas, interações, processos e ferramentas. O Scrum Guide define o objetivo das Retrospectivas

como determinar "como foi o último Sprint" em relação tanto às pessoas que fazem o trabalho, seus relacionamentos, o processo empregado e as ferramentas utilizadas (MATTHIES & DOBRIGKEIT, 2021).

As retrospectivas são uma realização do princípio "inspecionar e adaptar" dos métodos de desenvolvimento de software Ágil (MATTHIES & DOBRIGKEIT, 2021). Dessa forma, essa perspectiva iterativa e evolutiva contrasta com outros modelos preditivos tradicionais e é central para garantir que a equipe possa reagir rapidamente a mudanças e aprender com as experiências anteriores.

Em suma, metodologias ágeis como o Scrum trazem a melhoria contínua para o centro do desenvolvimento de software, institucionalizando práticas (como a retrospectiva) que estimulam times a revisar seu modo de trabalho regularmente e a buscar formas de torná-lo mais eficiente e eficaz. Retrospectivas de nível de equipe são amplamente utilizadas no desenvolvimento de software ágil e Lean (LEHTINEN et al., 2017). Essa filosofia estabelece o contexto e a motivação para a criação de ferramentas que apoiem e aprimorem ainda mais tal processo de aprendizagem.

2.2 - RETROSPECTIVAS DE SPRINT

Entre os eventos do Scrum, a Retrospectiva de Sprint destaca-se como o principal mecanismo formal para promoção da melhoria contínua dentro da equipe (DERBY & LARSEN, 2006). Segundo a definição do Guia do Scrum, "a Retrospectiva da Sprint é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint" (SCHWABER & SUTHERLAND, 2017, p. 13). Em outras palavras, trata-se de uma reunião realizada ao final de cada sprint na qual a equipe reflete sobre como o trabalho foi executado – examinando pessoas, processos, ferramentas e interações – e identifica ações concretas para aprimorar seu modo de trabalho no ciclo seguinte.

O propósito da Retrospectiva da Sprint é inspecionar como a última Sprint foi em relação às pessoas, aos relacionamentos, aos processos e às ferramentas; identificar e ordenar os principais itens que foram bem e as potenciais melhorias; e criar um plano para implementar melhorias no modo que o Time Scrum faz seu

trabalho (SCHWABER & SUTHERLAND, 2017, p. 14). Diferentemente da Reunião de Revisão (que foca o que foi desenvolvido, inspecionando o incremento do produto, voltada mais para o cliente), a Retrospectiva foca como o desenvolvimento aconteceu, abrindo um espaço seguro para discussão franca sobre o que ocorreu de positivo e o que não saiu conforme o esperado durante a iteração.

2.3.1 - IMPORTÂNCIA DAS RETROSPECTIVAS PARA EQUIPES ÁGEIS

A importância das retrospectivas de sprint para o sucesso de equipes ágeis é amplamente reconhecida na literatura. Derby e Larsen (2006) ressaltam que retrospectivas efetivas contribuem para aumento da produtividade da equipe, melhoria da qualidade do produto e maior satisfação dos membros, pois permitem resolver problemas e incentivar a experimentação de novas práticas em curtos ciclos de feedback. Ao proporcionar um momento dedicado de reflexão e ajuste, as retrospectivas atuam como catalisadoras da aprendizagem organizacional contínua.

Estudos empíricos confirmam esses benefícios: equipes que realizam retrospectivas regularmente tendem a identificar e remover impedimentos mais rapidamente, melhorar suas estimativas e aprimorar a colaboração interna (ERDOĞAN et al., 2018). Um estudo empírico específico analisou 109 issues e 36 action items de retrospectivas em projetos ágeis de grande escala, demonstrando quantitativamente os benefícios dessas práticas (Learning in the Large, 2018).

Além disso, a retrospectiva promove mais segurança do time ao incentivar que todos os membros – Desenvolvedores, Testados, Product Owner e Scrum Master – expressem abertamente suas perspectivas sobre o processo, em um ambiente de respeito e confiança mútua (MARSDEN et al., 2021). Para maximizar esse potencial, práticas como a Prime Directive (diretriz proposta por Norm Kerth que estabelece um ambiente sem julgamentos) costumam ser adotadas, assegurando que o foco esteja em aprimorar o processo e não em atribuir culpas.

2.3.2 - ESTRUTURA E PRÁTICAS DAS RETROSPECTIVAS

Em termos de estrutura, Derby e Larsen (2006) descrevem uma sequência típica para conduzir retrospectivas de forma eficaz: preparação do ambiente (set the stage), coleta de dados sobre a iteração (fatos marcantes, métricas, feedbacks), geração de insights através do debate do que foi bem-sucedido e o que pode melhorar, definição de ações de melhoria prioritárias e, por fim, o fechamento da retrospectiva com recapitulação dos compromissos assumidos pelo time.

Abordagens inovadoras têm sido propostas para aprimorar a eficácia das retrospectivas. Pesquisas recentes sobre abordagens baseadas em jogos demonstram como técnicas de gamificação podem aumentar o engajamento e a produtividade das sessões (Game-based Sprint Retrospectives, 2021).

Adicionalmente, estudos propõem o uso de dados de repositórios de software para informar retrospectivas ágeis, complementando percepções subjetivas das equipes com informações objetivas (Mining for Process Improvements, 2020).

O uso de técnicas variadas – como start, stop, continue, cinco porquês, diagramas de causa e efeito, dentre outras – pode tornar as sessões mais interativas e produtivas (OHNO, 1988).

2.3.3- COMPROMISSO COM A MELHORIA CONTÍNUA

Ao término de cada retrospectiva, espera-se que a equipe tenha elencado pelo menos algumas medidas de melhoria a serem implementadas no sprint seguinte, reforçando assim o ciclo de inspeção e adaptação contínuas do Scrum. Como estabelece o Guia do Scrum: "Ao final da Retrospectiva da Sprint, o Time Scrum deverá ter identificado melhorias que serão implementadas na próxima Sprint" (SCHWABER & SUTHERLAND, 2017, p. 14).

Para garantir que essas melhorias sejam efetivamente implementadas, "é incluído no mínimo um item de prioridade alta sobre melhoria do processo identificado na última Reunião de Retrospectiva" no Backlog da Sprint (SCHWABER & SUTHERLAND, 2017, p. 16).

Esse compromisso com a melhoria constante, sprint a sprint, é frequentemente referido como o kaizen do Scrum – uma aplicação do conceito japonês de aperfeiçoamento contínuo incremental no contexto de desenvolvimento ágil (IMAI, 1986). Essa filosofia estabelece as retrospectivas como um elemento fundamental para a evolução contínua das equipes e processos ágeis.

2.3 - FERRAMENTAS DE APOIO À MELHORIA CONTÍNUA

A mentalidade de melhoria contínua permeia as práticas ágeis, e diversas ferramentas e técnicas dão suporte a esse princípio durante o desenvolvimento de software. A busca por "melhoria contínua dentro da equipe de desenvolvimento" (SPICHKOVA et al., 2025, p. 1) é fundamental para o sucesso de equipes ágeis, que devem constantemente refletir sobre seus processos e práticas.

Uma das abordagens clássicas é o ciclo PDCA (Plan-Do-Check-Act), introduzido por Walter Shewhart e difundido por Deming na gestão da qualidade. No contexto ágil, cada iteração pode ser vista como um pequeno PDCA: planejar (Sprint Planning), fazer (execução do sprint), verificar (Sprint Review e métricas do sprint) e agir (Sprint Retrospective definindo ajustes) – instaurando um loop de melhoria a cada ciclo (POPPENDIECK & POPPENDIECK, 2003). A implementação de práticas de "engenharia de software contínua" demonstra como o ciclo PDCA pode ser aplicado sistematicamente no desenvolvimento ágil (FITZGERALD & STOL, 2017). A Retrospectiva da Sprint, em especial, corresponde às etapas de Check e Act do PDCA, ao inspecionar os resultados do sprint e estabelecer ações para aprimoramento no próximo ciclo.

Uma das ferramentas mais valiosas no processo de melhoria contínua é a análise de causa raiz dos problemas identificados. A Análise de Causa Raiz (RCA) é "uma investigação estruturada de um problema para detectar quais causas subjacentes precisam ser resolvidas" (LEHTINEN et al., 2014). Além da RCA, técnicas específicas como os 5 Porquês ajudam o time a investigar profundamente as causas subjacentes de falhas ou impedimentos. Ao "constantemente perguntar 'por quê'" para cada problema apontado, a equipe pode

descobrir questões sistêmicas e endereçá-las de forma mais efetiva (LEHTINEN et al., 2014).

Ferramentas visuais de qualidade, como o Diagrama de Ishikawa (diagrama de espinha de peixe), também podem ser empregadas em retrospectivas para organizar fatores contributivos de problemas. O "fishbone diagram" facilita a identificação de pontos de melhoria no processo (LEHTINEN et al., 2014). Estas técnicas de RCA podem ser utilizadas em retrospectivas ágeis: "cinco porquês, diagramas de espinha de peixe" são mencionados como técnicas de retrospectiva ágil (LEHTINEN et al., 2017).

No contexto da gestão ágil e Lean, a filosofia Kaizen merece destaque. Kaizen, que significa "melhoria contínua" em japonês, prega que pequenos aperfeiçoamentos diários podem resultar em grandes ganhos de eficiência e qualidade ao longo do tempo (IMAI, 1986). A filosofia de melhoria contínua tem suas raízes no pensamento Lean e enfatiza que mudanças pequenas e incrementais podem levar a melhorias significativas. Para viabilizar o kaizen no desenvolvimento de software, times ágeis frequentemente mantêm registros estruturados de ações de melhoria. Uma prática comum é registrar "propostas de ação" como resultado de retrospectivas (LEHTINEN et al., 2017), onde ficam documentadas as sugestões e lições aprendidas.

A ferramenta ARCA-tool, por exemplo, "colore as causas que têm ações corretivas" e permite monitoramento do status das ações: "ideia / será implementada / implementada" (LEHTINEN et al., 2014). Essa abordagem garante que iniciativas de aprimoramento não se percam e sejam tratadas com a devida importância, seguindo os princípios do "Kaizen-Kata" que estabelece rotinas sistemáticas para melhoria contínua (SUÁREZ-BARRAZA et al., 2012).

Existe uma variedade significativa de ferramentas de software concebidas para apoiar equipes na condução de retrospectivas e no acompanhamento de melhorias. Ferramentas online de retrospectiva fornecem quadros virtuais onde membros do time podem colaborar efetivamente. Ferramentas como "Miro e plataformas Ágeis como Jira" são amplamente utilizadas, assim como recursos como o site Retromat (MILANI et al., 2024, p. 2). A diversidade de opções

incluem "FigJam; Miro; RemoteRetro; EasyRetro; Retrium" (MILANI et al., 2024, p. 4), e também "Miro, TeamRetro, ou Atlassian Retrospective" (SPICHKOVA et al., 2025, p. 2).

Ao comparar as ferramentas disponíveis no mercado, nota-se que vem evoluindo com propostas inovadoras que endereçam lacunas específicas não cobertas pelas ferramentas generalistas apresentadas na Tabela 1. Nesse contexto, propomos o BugAwareRetro, uma solução especializada em retrospectivas com foco em defeitos e análise de bugs. Essa ferramenta integra automaticamente dados de defeitos ao Azure DevOps, com possibilidade de extensão para outras, categoriza causas raiz (requisitos, código, processo, ambiente), realiza análise entre feedback qualitativo e métricas de bugs e permite comparação de problemas entre sprints, preenchendo lacunas teóricos e práticos mapeados neste estudo.

Tabela 1 - Análise Comparativa das Ferramentas de Retrospectiva Existentes

Ferramenta	**Integração Jira**	**Integração Azure DevOps**	**Comparação entre Problemas**	**Preço**
EasyRetro	Sim - Jira Cloud apenas - Exportação de cards como issues - Escolha de projeto e tipo de issue	Não disponível	Não	Gratuito com limitações Planos pagos disponíveis
Retrium	Sim - Jira Cloud - Exportação de action items - Atribuição de usuários e datas	Não disponível	Não	Planos pagos

Parabol	Sim - Jira Cloud - Integração bidirecional - Importação de stories para estimativa - Exportação de tasks	Sim - Em beta preview - Exportação de tasks - Integração bidirecional para estimativas	Não	Gratuito até 2 times $6/usuário ativo/mês
ScatterSpoke	Sim - Integração básica	Não disponível	Não	Gratuito até 15 usuários $6/mês por usuário
Reetro	Não disponível	Não disponível	Não	Gratuito permanentemente
FunRetro	Não disponível	Não disponível	Não	Gratuito
Miro	Sim - Integração robusta - Sincronização com Jira - Criação de cards como tasks	Sim - Integração disponível - Sincronização de tasks - Criação de work items	Sim - AI clustering - Agrupamento automático - Análise de sentimento	Gratuito com limitações Planos pagos disponíveis

TeamRetro	Sim - Jira Cloud - Integração com Jira - Criação de action items	Sim - Integração com Azure DevOps - Criação de work items	Sim - Reporting e insights - Dashboards históricos - Análise de tendências	$20.83/mês (Single Team) $50/mês (Small Org) $75/mês (Large Org)
Atlassian Retrospective	Sim - Nativo no Jira - Kanban boards para retros - Criação de issues diretamente	Não disponível	Limitado - Histórico básico - Visualização de dados	Incluído no Jira Sem custo adicional
FigJam	Sim - Via marketplace - Embed de designs - Criação de tasks via widget	Sim - Via plugins - Exportação de componentes - Criação de work items	Não - Funcionalidade básica - Sem análise histórica	Incluído no Figma Gratuito básico
ARCA-tool	Não disponível	Não disponível	Sim - Análise longitudinal - Combinação de retrospectivas - Monitoramento de saídas	Gratuito Open-source (MIT)
BugAwareRetro	Sim - Jira Cloud e Server - Importação automática de defeitos - Categorização por causa raiz - Análise correlacional	Sim - Integração completa - Importação de work items - Análise de métricas de qualidade	Sim - Comparação entre sprints - Identificação de padrões recorrentes - Tendências de defeitos	Modelo freemium

Fonte: Os autores

O uso dessas plataformas digitais tem se tornado cada vez mais relevante, especialmente em equipes distribuídas ou remotas. A comparação de "35 ferramentas online de RCA para retrospectivas distribuídas" demonstra a amplitude de opções disponíveis (LEHTINEN et al., 2014, p. 4). Essas ferramentas incentivam a participação de todos de forma igualitária e podem fornecer recursos importantes como anonimato e votação em itens.

Estudos apontam que o emprego de ferramentas digitais com recursos de anonimização aumenta a segurança psicológica nas retrospectivas. "A ferramenta protege o anonimato dos membros da equipe" (LEHTINEN et al., 2014, p. 7), o que é fundamental para encorajar feedbacks mais honestos e construtivos. Uma retrospectiva bem-sucedida depende de "honestidade, segurança psicológica, abertura e um compromisso com a melhoria" (MILANI et al., 2024, p. 2). Criar um "ambiente psicologicamente seguro" é essencial (SPICHKOVA et al., 2025, p. 1-3).

Muitas dessas ferramentas permitem o armazenamento do histórico de retrospectivas, possibilitando análises longitudinais valiosas. A ferramenta ARCA-tool "fornece recursos para monitorar a saída de retrospectivas" e permite "análise de uma retrospectiva individual bem como a combinação de muitas retrospectivas" (LEHTINEN et al., 2014, p. 8-9). Um estudo longitudinal analisando "dados de 37 retrospectivas em nível de equipe por quase 3 anos" demonstra como essa análise temporal pode revelar "como os tópicos de discussão evoluem ao longo do tempo" (LEHTINEN et al., 2017, p. 2413-2414).

Vale mencionar que a própria mentalidade enxuta (Lean) trouxe para o desenvolvimento de software diversos conceitos de melhoria contínua suportados por ferramentas específicas. Um exemplo é o Kanban, que embora seja um método de trabalho em si, também funciona como ferramenta de visualização e otimização de fluxo. O "Kanban oferece uma abordagem evolutiva para mudança organizacional" e pode ser integrado com práticas Scrum existentes (ANDERSON, 2010). Equipes Scrum frequentemente incorporam práticas de Kanban, como quadros de tarefas e limites de WIP (trabalho em progresso), para monitorar e melhorar o fluxo de trabalho dentro do sprint.

Essas práticas ajudam a identificar gargalos e desperdícios (atividades que não agregam valor), alinhadas ao princípio Lean de eliminação de desperdício. A combinação de "Kanban e Scrum" permite que equipes obtenham "o melhor dos dois mundos" ao aplicar visualização de fluxo dentro de estruturas iterativas (KNIBERG & SKARIN, 2010). A aplicação de tecnologias como RFID em sistemas Kanban demonstra como ferramentas digitais podem apoiar a melhoria contínua em ambientes de desenvolvimento complexos (LEI et al., 2017).

Em suma, as metodologias ágeis fazem amplo uso de instrumentos de apoio que mantêm o foco na melhoria contínua dos processos de desenvolvimento. Através de ciclos estruturados (PDCA), técnicas investigativas como RCA e 5 Porquês, conceitos filosóficos (Kaizen), ferramentas digitais de colaboração que promovem segurança psicológica, e sistemas de visualização de fluxo (Kanban), as equipes podem institucionalizar um processo robusto de aprendizado e evolução contínua.

2.4 - ANÁLISE DE DADOS NO DESENVOLVIMENTO ÁGIL

Com a maturidade das abordagens ágeis, tem crescido a ênfase em tomar decisões orientadas por dados (data-driven) durante o desenvolvimento de software. A análise de dados no contexto ágil refere-se tanto ao monitoramento de métricas do processo de desenvolvimento quanto ao uso de informações empíricas para retroalimentar e melhorar a gestão do projeto e a qualidade do produto. "Métricas são elementos-chave que podem nos fornecer informações valiosas sobre a eficácia dos processos de desenvolvimento de software ágil, particularmente considerando o ambiente Scrum" (ALMEIDA & CARNEIRO, 2023, p. 1). Diferentemente dos métodos tradicionais, nos quais extensos relatórios e indicadores eram coletados ao final de um projeto, nas metodologias ágeis a coleta e interpretação de métricas acontece de forma contínua e em tempo real, a cada iteração. Isso permite ajustes rápidos de curso e baseados em evidências, reforçando o princípio de inspeção e adaptação do Agile.

Métricas Ágeis típicas incluem, por exemplo, a velocidade da equipe, definida como "quantidade de trabalho que uma equipe de desenvolvimento pode

fazer durante um sprint" (ALMEIDA & CARNEIRO, 2023, p. 4), o Lead Time e Cycle Time (tempos decorridos desde o início até a conclusão de um item, ou de uma etapa do fluxo) (PETERSEN & WOHLIN, 2019), o Burndown Chart, caracterizado como "representação gráfica da taxa em que o trabalho é completado e quanto trabalho resta para ser realizado em um sprint" (ALMEIDA & CARNEIRO, 2023, p. 4), a cumulative flow diagram (fluxo acumulado de tarefas em diferentes estados) (AHMAD et al., 2018, p. 89), a taxa de débitos técnicos ou "densidade de defeitos: número de defeitos encontrados dividido pelo tamanho do módulo/software considerado" (ALMEIDA & CARNEIRO, 2023, p. 4), e a "porcentagem de automação de testes considerando testes automáticos e manuais" (ALMEIDA & CARNEIRO, 2023, p. 4).

Cada uma dessas métricas fornece um vislumbre de aspectos da performance do time ou qualidade do produto, e a combinação inteligente delas pode oferecer percepções valiosas. Por exemplo, ao analisar a variação da velocidade ao longo de múltiplos sprints, a equipe pode identificar tendência de melhora (aprendizado e otimização) ou queda (possíveis problemas sistêmicos) em sua produtividade. Da mesma forma, um gráfico de ciclo de vida das issues (controle de fluxo) pode revelar gargalos – se muitas tarefas ficam acumuladas em "Em andamento" por tempo excessivo, pode indicar impedimentos ou sobrecarga em determinado estágio. Estudos indicam que "indivíduos com mais anos de experiência têm uma percepção maior da importância de métricas relacionadas ao desempenho da equipe" (ALMEIDA & CARNEIRO, 2023, p. 8), utilizando-as para direcionar melhorias no processo.

Um aspecto crucial da análise de dados em ambientes ágeis é empregá-la como insumo nas retrospectivas e no planejamento. Erdoğan et al. (2018) demonstraram que "dados históricos do projeto, quando coletados e analisados adequadamente, podem tornar as retrospectivas ainda mais eficazes, provendo evidências objetivas para embasar as discussões do time" (referenciado em ALMEIDA & CARNEIRO, 2023, p. 3). Nesse estudo, os autores avaliam quais tipos de dados são úteis para monitorar e guiar melhorias, e propõem a coleta regular de estatísticas como: correlação entre estimativas (conhecidas como story

points) e esforço real de cada item (para avaliar a acurácia das estimativas e fomentar ajustes na técnica de estimativa); velocidade do time vs. esforço não planejado (para verificar se trabalho emergencial está prejudicando as entregas); taxa de defeitos por componente (para identificar módulos problemáticos que demandam refactoring ou mais testes); e esforço alocado em atividades de qualidade (porcentagem do esforço do sprint investido em testes, revisões de código, etc.) (ERDOĞAN et al., 2018).

A inclusão dessas análises quantitativas nas retrospectivas fornece ao time uma visão empiricamente embasada de seu desempenho e das causas de eventuais desvios, alinhando-se ao conceito de melhoria baseada em evidências. Como resultado, as equipes podem definir ações de melhoria mais precisas – por exemplo, ajustar a técnica de estimativa se for constatado um desalinhamento sistemático, incrementar a cobertura de testes em áreas com alta densidade de defeitos, ou redistribuir a carga de trabalho se certos membros estiverem constantemente sobrecarregados.

Ferramentas de Business Intelligence e DevOps Analytics têm emergido para auxiliar na coleta e visualização dessas métricas ágeis (FITZGERALD & STOL, 2017). Muitas suites ALM (Application Lifecycle Management) e plataformas como Jira, Azure DevOps e outras, já oferecem dashboards integrados que compilam dados do repositório de código, do sistema de issues e do pipeline de integração contínua, apresentando indicadores-chave em tempo real (CHEN, 2020, p. 156). Por exemplo, é possível visualizar instantaneamente o burndown de uma sprint, a taxa de sucesso dos builds ou implantações (deployment frequency), e até indicadores de satisfação do cliente (como o NPS ou feedbacks coletados de revisões).

A literatura recente reforça que "a visualização de métricas em tempo real contribui para a transparência e para a tomada de decisão rápida dentro das equipes ágeis" (referenciado em ALMEIDA & CARNEIRO, 2023, p. 3). Não obstante, é fundamental que as métricas sejam usadas de forma construtiva: o objetivo não é microgerenciar ou punir desempenhos abaixo da meta, mas sim oferecer um sistema de alerta antecipado e um termômetro da saúde do projeto

para que o próprio time possa se auto-organizar em busca de melhoria contínua (referenciado em ALMEIDA & CARNEIRO, 2023).

A análise de dados no desenvolvimento ágil transcende métricas processuais, abrangendo também o aprendizado sistemático a partir de informações de uso do produto em ambiente produtivo. Metodologias contemporâneas como DevOps e Lean Startup incorporam fundamentalmente o ciclo construir-medir-aprender, no qual dados operacionais reais incluindo telemetria, métricas de negócio e feedback de usuários são analisados continuamente para orientar a evolução estratégica do produto (HUMBLE & FARLEY, 2019).

Esta convergência entre desenvolvimento ágil e análise de dados de produto possibilita ajustes rápidos de estratégias e funcionalidades baseados em evidências concretas de valor ou necessidade do cliente. Portanto, a cultura ágil contemporânea valoriza não apenas a capacidade de resposta rápida a mudanças, mas também a resposta estratégica fundamentada em dados empíricos, seja para aprimoramento dos processos internos de desenvolvimento, seja para maximização do valor entregue ao usuário final.

3 - BUGAWARERETRO: FERRAMENTA DE APOIO À GESTÃO DE BUGS NAS REUNIÕES DE RETROSPECTIVA

Este capítulo descreve a metodologia adotada no desenvolvimento do BugAwareRetro, incluindo: visão geral, levantamento dos requisitos funcionais e não funcionais, casos de uso, diagrama de classes e diagrama de sequência. Apresentam-se, em seguida, os artefatos elaborados e justificativas técnicas para as escolhas realizadas, fundamentadas em práticas consolidadas da Engenharia de Software.

3.1 - VISÃO GERAL

A metodologia adotada fundamenta-se em uma abordagem que busca desenvolver soluções práticas para problemas reais através de artefatos tecnológicos validados empiricamente (HEVNER et al., 2004). Esta abordagem em Engenharia de Software enfatiza a importância da construção de artefatos que não apenas resolvam problemas específicos, mas também contribuam para o avanço do conhecimento científico na área, caracterizando-se como "um paradigma de pesquisa pragmático, abordando características de pesquisa aplicada e prescritiva" (WIERINGA, 2014, p. 23-45).

O BugAwareRetro constitui-se como uma ferramenta orientada a dados derivados de atividades de teste, alinhando-se com os princípios do Test-Driven Development (TDD) e Behavior-Driven Development (BDD), onde a documentação de falhas constitui elemento central para melhoria contínua (BECK, 2003; NORTH, 2007). A integração entre dados de teste e retrospectivas permite análise mais precisa dos padrões de defeitos, facilitando a identificação de causas raiz baseadas em evidências empíricas.

Ferramentas existentes como Jira, Azure DevOps e GitLab Issues oferecem funcionalidades isoladas de rastreamento de bugs ou gestão de retrospectivas, porém carecem de integração sistêmica entre ambos os domínios. Esta fragmentação resulta em perda de contexto e dificuldades na correlação entre defeitos identificados e ações de melhoria propostas nas retrospectivas (RODRÍGUEZ et al., 2022). O BugAwareRetro diferencia-se ao propor uma

solução integrada que unifica estes domínios, permitindo análises correlacionais entre tipos de defeitos e eficácia das ações implementadas.

O Sistema BugAwareRetro enquadra-se adequadamente neste paradigma ao constituir um artefato tecnológico que aborda problemas relevantes do desenvolvimento ágil, especificamente a integração entre rastreabilidade de bugs e gestão de retrospectivas. O projeto atende aos critérios fundamentais de DSR através de contribuições claras em construtos, métodos e instanciação, evidenciando um processo iterativo de refinamento e documentação técnica rigorosa. Contudo, para consolidar completamente sua aderência aos princípios de DSR, o projeto necessita de métodos de avaliação empírica mais robustos para demonstrar a eficácia do artefato em ambientes organizacionais reais, bem como de fundamentação teórica mais explícita das decisões de design adotadas, aspectos essenciais para validação científica segundo os critérios estabelecidos por Hevner et al. (2004).

A aplicação da DSR no contexto de Engenharia de Software permite melhorar a comunicação das contribuições de pesquisa, especialmente em projetos que visam fornecer valor prático à indústria (PEFFERS et al., 2020, p. 1-18). Estudos empíricos demonstram que "a lente da design science ajuda a identificar a contribuição teórica de uma saída de pesquisa, que por sua vez é o núcleo para avaliar a relevância prática e novidade da regra prescrita" (JOHANNESSON & PERJONS, 2019, p. 89-123).

O objetivo central desta fase é articular o desenvolvimento do backend (Java-Spring Boot) e frontend (React-TypeScript), evidenciando as técnicas empregadas em ambos os lados. A escolha pela arquitetura em camadas com separação clara entre frontend e backend fundamenta-se nos princípios da arquitetura orientada a serviços (SOA) e nos padrões de desenvolvimento web modernos (FIELDING & TAYLOR, 2002, p. 115-150).

No backend, implementaram-se serviços de domínio e repositórios para persistência e lógica de negócio, seguindo padrões arquiteturais que promovem a separação clara de responsabilidades e facilitar a manutenibilidade do sistema. A

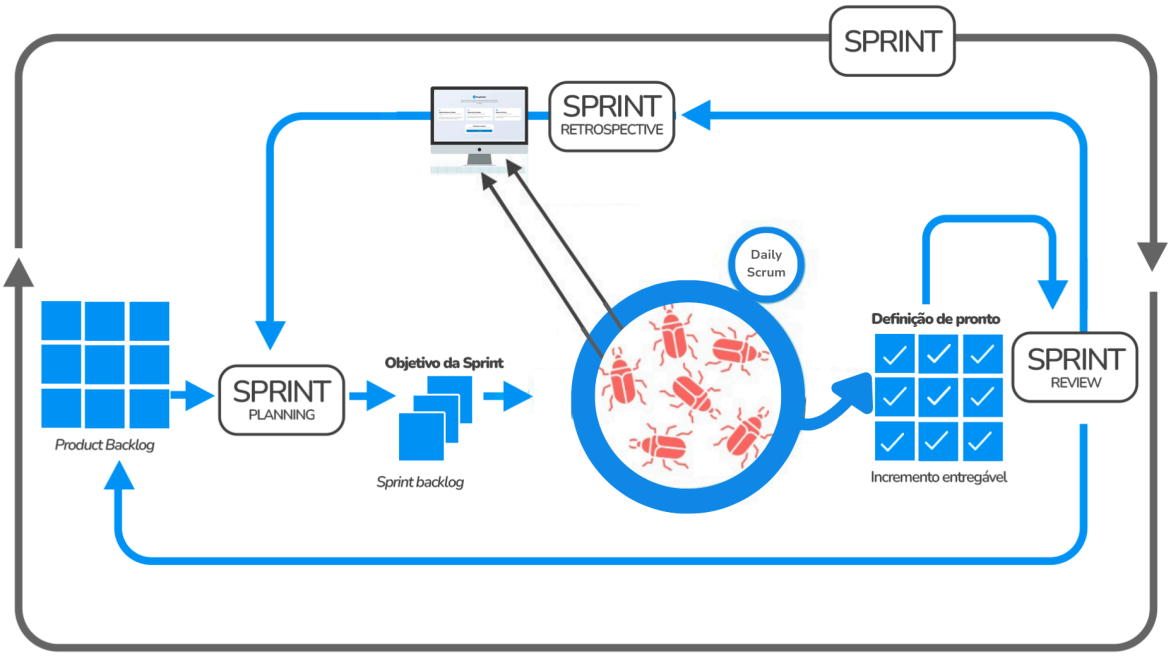
arquitetura adotada permite otimizar as operações de consulta e manipulação de dados, contribuindo para melhor desempenho e escalabilidade (FOWLER, 2011).

No frontend, desenvolveu-se o dashboard interativo e formulários de captura de feedback, utilizando componentes reutilizáveis, hooks e React Query para sincronização de estado. A adoção de componentes reutilizáveis segue os princípios da engenharia de software baseada em componentes (Component-Based Software Engineering - CBSE), que promove a modularidade, reutilização e manutenibilidade (SZYPERSKI, 2002). O React Query implementa o padrão de gerenciamento de estado servidor-cliente, facilitando a sincronização de dados e melhorando a experiência do usuário (OSMANI, 2017).

Esta abordagem sistemática permitiu a validação contínua dos requisitos implementados, assegurando que o artefato final atendesse aos objetivos propostos e aos critérios de qualidade estabelecidos na metodologia. A implementação dos cenários previstos contribuiu significativamente para a validação técnica do sistema, elemento fundamental para a avaliação da eficácia do artefato desenvolvido.

A Figura 1 apresenta um esquema que demonstra a integração do sistema BugAwareRetro na metodologia Scrum, atuando em dois momentos específicos: durante a Sprint, quando os defeitos e falhas identificados são registrados no sistema como repositório central para documentação de problemas; e na Cerimônia de Retrospectiva, quando o BugAwareRetro retorna à atividade apresentando de forma organizada todos os dados previamente mapeados, disponibilizando os registros coletados para análise e discussão pela equipe, garantindo assim um ciclo contínuo de identificação, registro e revisão de problemas no processo de desenvolvimento.

Figura 1- Esquema de contexto do BugAwareRetro



Fonte: Os autores

3.2 - LEVANTAMENTO DE REQUISITOS

A etapa de definição da lista de requisitos envolveu a elaboração de casos de uso e especificações funcionais que consolidam os requisitos identificados para o sistema. A importância da documentação estruturada de requisitos é amplamente reconhecida na literatura de Engenharia de Software como fundamental para o sucesso de projetos (SOMMERVILLE, 2015). A modelagem de casos de uso serve como artefato central para comunicação entre stakeholders e como base para validação e verificação do sistema desenvolvido (WIEGERS & BEATTY, 2013). No contexto específico de sistemas para retrospectivas ágeis, a literatura evidencia que a maioria das atividades propostas para a fase de coleta de dados carecem de conexões explícitas com dados objetivos de projeto, baseando-se predominantemente nas percepções dos participantes (MATTHIES; DOBRIGKEIT, 2021).

A técnica de levantamento de requisitos adotada combina análise documental, conversas informais estruturadas com profissionais da indústria e observação de processos existentes. Esta abordagem multi-método é recomendada

por Robertson e Robertson (2012, p. 123-145) para garantir a completude e precisão dos requisitos identificados. Pesquisas realizadas recentemente apontam que, apesar de equipes rotineiramente coletem dados de projeto através de alguma ferramenta de de controle de versão, rastreamento de problemas e métricas de desenvolvimento, elas raramente utilizam essas informações efetiva e sistematicamente durante as retrospectivas (MATTHIES; HESSE, 2019). A integração efetiva de dados objetivos em retrospectivas é reconhecida como um princípio fundamental para tomada de decisões baseada em evidências, conforme estabelecido pela teoria de controle de processo empírico do Scrum (MATTHIES; HESSE, 2019).

Foram conduzidas conversas estruturadas com vinte e quatro profissionais atuantes em diferentes áreas do desenvolvimento de software, representando perspectivas diversificadas sobre os desafios relacionados ao rastreamento de defeitos e condução de retrospectivas ágeis. Além de abordar pontos relevantes na metodologia ágil e a percepção de cada papel diante da cerimônia de retrospectiva.

O grupo consultado incluiu três analistas de qualidade de software (QA) responsáveis pela identificação e documentação de defeitos, oito desenvolvedores de software especializados em desenvolvimento web com experiência em correção de bugs e implementação de funcionalidades, dois engenheiros de software, dois cientistas de dados focados em análise de métricas de desenvolvimento, dois arquitetos de software e um arquiteto de soluções especializados em design de sistemas e qualidade técnica, cinco profissionais da equipe de negócio compreendendo analistas de requisitos, product owners e gerentes de produto, além de dois UX/UI designers responsáveis pela prototipagem de interfaces e experiência do usuário. Durante as entrevistas, confirmou-se que as principais barreiras para o uso de dados em retrospectivas incluem questões com segurança psicológica, desconexão entre coleta de dados e integração real nas reuniões, e preferência por métodos simples de rastreamento em detrimento de abordagens analíticas mais avançadas (MILANI et al., 2025).

A análise de processos ágeis existentes, particularmente das cerimônias de retrospectiva, forneceu percepções valiosas sobre as necessidades reais dos usuários e as limitações das ferramentas atuais (DERBY & LARSEN, 2006). Alguns estudos indicam que aproximadamente 86% das atividades de coleta de dados em retrospectivas não fazem menção ou não aproveitam dados de projeto de software, baseando-se unicamente nas opiniões, feedbacks e percepções das pessoas da equipe (MATTHIES; DOBRIGKEIT, 2021). A literatura aponta que retrospectivas efetivas devem começar com dados objetivos e reais, incluindo eventos da iteração, métricas coletadas e funcionalidades ou user stories finalizadas, para garantir uma visão mais completa para todos os membros da equipe (KERTH, 2001).

Esta diversidade de perfis profissionais proporcionou compreensão abrangente dos workflows de desenvolvimento, desde a identificação inicial de defeitos até sua resolução e impacto nas cerimônias de retrospectiva, evidenciando lacunas significativas nas ferramentas existentes, particularmente relacionadas à correlação entre dados quantitativos de defeitos e percepções qualitativas coletadas durante retrospectivas. A pesquisa revela ainda que, embora a maioria das equipes entendam que as retrospectivas têm potencial para melhorar a colaboração da equipe, existe uma percepção variada sobre seu impacto no trabalho efetivamente realizado, sugerindo necessidade de abordagens mais equilibradas entre reflexões centradas no aspecto humano e insights baseados em dados (MILANI et al., 2025).

3.2.1 - REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem funcionalidades essenciais como: cadastro de bugs, tarefas e histórias de usuário; coleta de feedback dos membros da equipe via questionários; geração de relatórios de análise por sprint (gráficos, indicadores de defeitos, velocidade); exportação de dados; e gerenciamento de sessões de retrospectiva com registro de ações definidas. A identificação e especificação de requisitos funcionais seguiram as diretrizes do IEEE Standard

830-1998 para documentação de requisitos de software (IEEE, 1998). Cada requisito funcional foi especificado com critérios de aceitação claros, prioridade e rastreabilidade, garantindo que todas as funcionalidades essenciais fossem capturadas e validadas (DAVIS, 1993). A técnica de User Stories, amplamente utilizada em metodologias ágeis, foi empregada para capturar requisitos funcionais de forma centrada no usuário (COHN, 2004).

O cadastro de bugs, tarefas e histórias de usuário representa um conjunto fundamental de funcionalidades que permite a rastreabilidade de itens de trabalho ao longo do ciclo de desenvolvimento. Esta funcionalidade baseia-se nos princípios da gestão de configuração de software e rastreabilidade de requisitos (PRESSMAN & MAXIM, 2020). A coleta de feedback via questionários estruturados fundamenta-se em técnicas de pesquisa quantitativa e qualitativa, permitindo a captura sistematizada de percepções da equipe sobre qualidade e processos (KITCHENHAM & PFLEEGER, 2008).

A seguir, apresenta-se a listagem de casos de uso organizados por módulos funcionais que representam as User Stories do sistema:

Gerenciamento de Retrospectivas

UC001 - Criar Sessão de Retrospectiva: Permite iniciar uma nova sessão de retrospectiva vinculada a uma sprint específica, definindo participantes e objetivos. Os atores envolvidos são Scrum Master e Product Owner.

UC002 - Configurar Dinâmica de Retrospectiva: Define a metodologia a ser utilizada e estabelece questões norteadoras para a sessão. O ator responsável é o Scrum Master.

UC003 - Coletar Feedback dos Participantes: Fornece interface estruturada para membros da equipe submeterem percepções categorizadas durante a retrospectiva. Os atores envolvidos são Developer, Scrum Master e Product Owner.

UC004 - Integrar Dados Quantitativos: Vincula automaticamente métricas de defeitos e indicadores de performance da sprint à sessão de retrospectiva ativa. O ator responsável é o Sistema.

UC005 - Gerar Análise Correlacional: Produz correlações entre feedback qualitativo dos participantes e dados quantitativos de defeitos e métricas da sprint. O ator responsável é o Sistema.

UC006 - Facilitar Discussão Colaborativa: Disponibiliza interface para discussão estruturada dos insights gerados, permitindo comentários e refinamentos. Os atores envolvidos são Scrum Master, Developer e Product Owner.

UC007 - Definir Ações de Melhoria: Documenta ações específicas de melhoria com responsáveis designados, prazos e critérios de sucesso. Os atores envolvidos são Scrum Master e Product Owner.

UC008 - Acompanhar Implementação de Ações: Monitora o progresso das melhorias definidas e seu impacto nas sprints subsequentes. Os atores envolvidos são Scrum Master e Sistema.

UC009 - Visualizar Histórico de Retrospectivas: Apresenta evolução das retrospectivas e efetividade das ações implementadas ao longo do tempo. Os atores envolvidos são Scrum Master e Product Owner.

Gerenciamento de Sprints

UC010 - Criar Nova Sprint: Permite a criação de uma nova sprint com objetivos, cronograma e critérios de sucesso definidos. Os atores envolvidos são Scrum Master e Product Owner.

UC011 - Visualizar Métricas da Sprint: Apresenta indicadores de performance detalhados para subsidiar análises retrospectivas. Os atores envolvidos são Scrum Master, Developer, Product Owner e Sistema.

UC012 - Finalizar Sprint com Análise: Encerra uma sprint ativa gerando automaticamente dados estruturados para a retrospectiva subsequente. Os atores envolvidos são Scrum Master e Sistema.

Gerenciamento de Defeitos

UC013 - Registrar Defeito Contextualizado: Cria registro detalhado de defeito incluindo contexto de ocorrência para análise retrospectiva. Os atores envolvidos são Developer e Scrum Master.

UC014 - Categorizar Causa Raiz: Classifica defeitos por origem (requisitos, código, processo, ambiente) facilitando análises de tendências. Os atores envolvidos são Developer e Scrum Master.

UC015 - Rastrear Ciclo de Vida do Defeito: Monitora automaticamente o tempo de detecção, resolução e impacto de cada defeito. O ator responsável é o Sistema.

UC016 - Importar Defeitos de Ferramentas Externas: Integra dados de ferramentas como Jira, Azure DevOps e similares via API. Os atores envolvidos são Sistema e Scrum Master.

Análise e Relatórios

UC017 - Gerar Dashboard de Retrospectivas: Apresenta métricas consolidadas e tendências das retrospectivas realizadas. Os atores envolvidos são Sistema e Scrum Master.

UC018 - Exportar Relatórios Estruturados: Gera relatórios executivos para gestão organizacional em formatos padronizados. Os atores envolvidos são Scrum Master e Product Owner.

3.2.2 - REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais incluem critérios de desempenho (tempo de resposta $\leq$ 500 ms para 95% das requisições), disponibilidade, segurança

(autenticação via OAuth2/JWT, validação de entrada e sanitização), escalabilidade, testabilidade, cobertura mínima de testes (≥ 80%) e usabilidade (interface responsiva com padrões de acessibilidade).

A especificação de requisitos não funcionais seguiu a taxonomia proposta por Chung et al. (2012, p. 15-45), que categoriza requisitos de qualidade em dimensões mensuráveis. O estabelecimento de critérios quantitativos para desempenho (tempo de resposta ≤ 500 ms) baseia-se em estudos de usabilidade que demonstram a importância da responsividade para a experiência do usuário (NIELSEN, 1993). A meta de 500 ms está alinhada com as diretrizes de performance web do W3C e com estudos empíricos sobre percepção de latência em aplicações web (SEOW, 2008).

Os requisitos de segurança, incluindo autenticação via OAuth2/JWT, refletem as melhores práticas de segurança para aplicações web modernas. O protocolo OAuth2 é amplamente reconhecido como padrão para autorização delegada em sistemas distribuídos (HARDT, 2012), enquanto JWT (JSON Web Tokens) fornece um mecanismo seguro e escalável para transmissão de informações de autenticação (JONES et al., 2015). A validação de entrada e sanitização de dados são práticas fundamentais para prevenção de vulnerabilidades de injeção de código e cross-site scripting (XSS) (HOWARD & LEBLANC, 2003).

3.3 - CASOS DE USO

A modelagem de casos de uso foi documentada em artefatos específicos que representam as interações essenciais do sistema: Scrum Master inicia sessões de retrospectiva baseadas em dados, desenvolvedores submetem feedback estruturado, o sistema correlaciona informações qualitativas e quantitativas, e a equipe define ações de melhoria fundamentadas em evidências concretas. Cada caso de uso contém descrição textual, atores, pré-condições, fluxo principal, fluxos alternativos e cenários de exceção.

A modelagem de interações entre esses atores e o sistema captura os workflows essenciais das cerimônias de retrospectiva aprimoradas, permitindo a sistematização de processos tradicionalmente baseados apenas em percepções subjetivas e transformando-os em análises fundamentadas em dados empíricos.

3.4 - DIAGRAMA DE CLASSES

A estrutura de classes define as entidades do domínio – como Bug, Sprint, Feedback, Indicador, Ação de Melhoria – e suas relações. O diagrama na Figura 2 apresenta agregados, value objects e entidades associadas aos módulos de domínio. Explica-se como cada classe representa conceitos essenciais da rastreabilidade e análise de defeitos, facilitando a compreensão de persistência, transição de estados e regras de negócio (por exemplo, ao atribuir um bug a uma pessoa ou calcular métricas de ciclo).

A entidade Bug representa o conceito central do domínio, encapsulando informações sobre defeitos identificados durante o desenvolvimento. A modelagem desta entidade incorpora atributos essenciais como identificação única, descrição, severidade, status, responsável e histórico de mudanças.

A entidade Sprint modela o conceito de iteração no desenvolvimento ágil, agregando informações sobre período, objetivos, equipe e métricas. Esta entidade serve como contexto agregador para bugs, tarefas e feedback, permitindo análises temporais e comparativas entre diferentes iterações. A modelagem da Sprint incorpora padrões de agregação que garantem a consistência transacional e facilitam consultas relacionadas a métricas de qualidade e performance da equipe (FOWLER, 2002). Os Value Objects identificados no domínio incluem conceitos como Métrica, Período e Status, que representam valores sem identidade própria, mas com significado semântico importante. A utilização de value objects segue as práticas recomendadas para garantir imutabilidade e comparabilidade por valor, reduzindo a complexidade do modelo e melhorando a expressividade do código (EVANS, 2023).

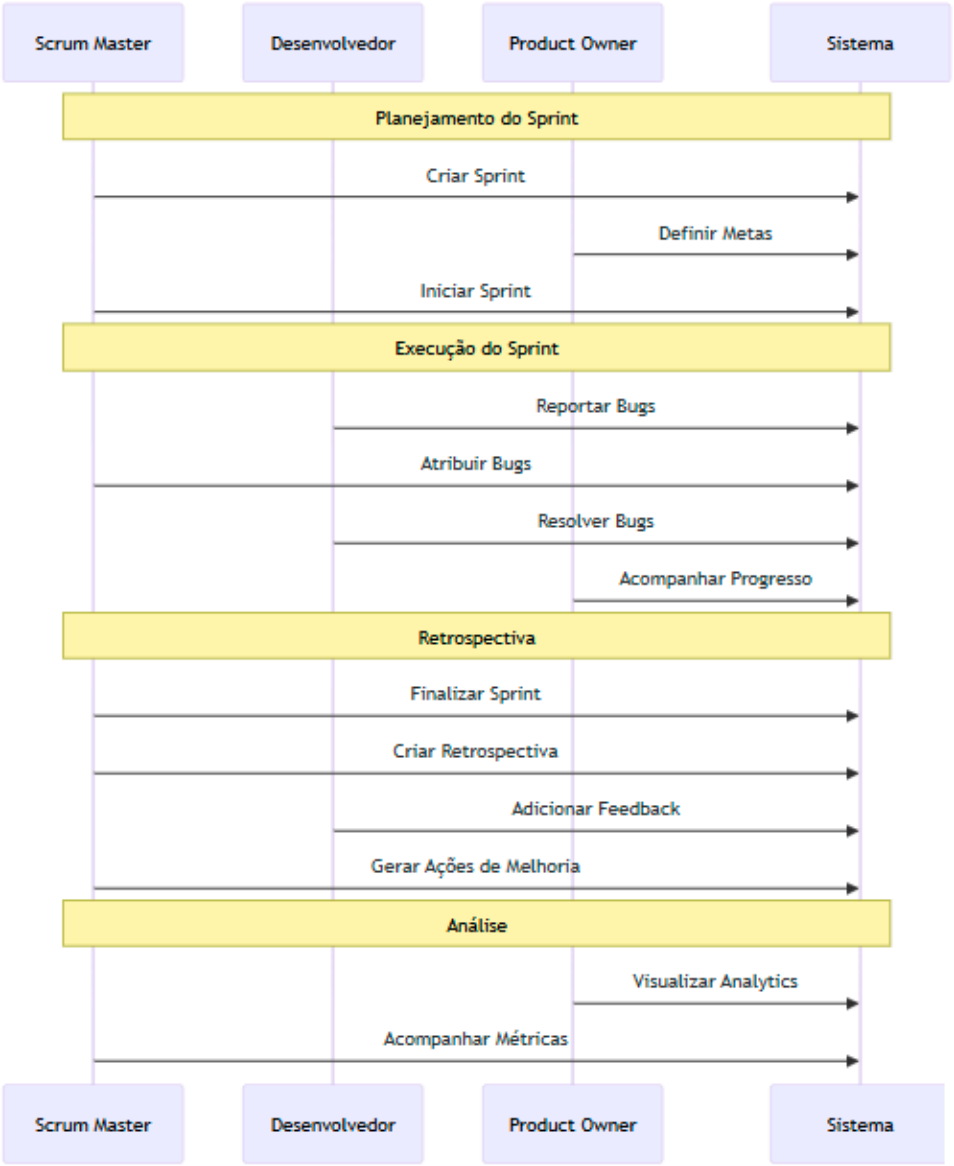
Devido à complexidade e à dimensão do diagrama de classes, sua representação completa encontra-se no Anexo A deste trabalho.

3.5 - DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência, conforme Figura 2, apresenta o ciclo completo de uma Sprint no sistema BugAwareRetro, demonstrando as interações fundamentais entre a interface React e os serviços Spring Boot. Inicialmente, durante a fase de planejamento, o Scrum Master executa o registro e inicialização da sprint no sistema, enquanto o Product Owner estabelece concomitantemente os objetivos estratégicos e define as metas a serem alcançadas durante a iteração. Logo após a conclusão do planejamento, inicia-se a fase de execução, onde o Desenvolvedor realiza sistematicamente o reporte, correção e monitoramento de defeitos identificados. Paralelamente, o Scrum Master conduz a triagem dos bugs reportados e acompanha o progresso da sprint em colaboração com o Product Owner, garantindo continuamente o alinhamento com os objetivos previamente estabelecidos.

Posteriormente, com a conclusão da Sprint, procede-se à fase de retrospectiva, iniciando-se com a finalização formal da iteração atual e abertura da retrospectiva propriamente dita. Durante esta etapa, realiza-se a coleta sistemática de feedbacks dos participantes e consolidam-se as ações de melhoria identificadas para implementação nas próximas iterações. Finalmente, na fase de análise, os stakeholders acessam painéis analíticos e métricas agregadas disponibilizadas pelo sistema, possibilitando uma avaliação objetiva e quantitativa da qualidade do software e da produtividade da equipe de desenvolvimento. Consequentemente, esta arquitetura de processo assegura a rastreabilidade completa do ciclo de desenvolvimento e fornece dados estruturados para tomada de decisões baseada em evidências.

Figura 2 - Diagrama de fluxo das funcionalidades internas



Fonte: Os autores

4 - DESENVOLVIMENTO DO SISTEMA

Este capítulo apresenta os aspectos técnicos e arquiteturais do desenvolvimento do Sistema BugAwareRetro, detalhando as tecnologias adotadas, padrões implementados e decisões de design que fundamentaram a construção da solução. A abordagem metodológica visa criar um artefato tecnológico robusto que atenda aos requisitos identificados e contribua para o aprimoramento das práticas de gestão de qualidade em desenvolvimento ágil de software.

4.1 - TECNOLOGIAS UTILIZADAS

4.1.1 - CAMADA DE BACKEND

A camada de backend foi desenvolvida utilizando o framework Spring Boot versão 3.4 em conjunto com Java 21, constituindo o núcleo da aplicação responsável pelo fornecimento da API REST para gerenciamento de dados relacionados a bugs, sprints e retrospectivas. Esta escolha tecnológica fundamenta-se na robustez, escalabilidade e facilidade de implementação de padrões arquiteturais enterprise que o ecossistema Spring proporciona, atendendo diretamente aos requisitos não funcionais de escalabilidade e manutenibilidade estabelecidos para o sistema.

O sistema de persistência de dados emprega o PostgreSQL como banco de dados relacional principal, escolhido por sua elevada confiabilidade, suporte robusto a transações compatíveis com o modelo ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e capacidade de escalabilidade tanto vertical (aumento dos recursos de hardware da máquina) quanto horizontal (por meio de replicação e arquiteturas distribuídas). A integração entre a aplicação e o banco de dados é realizada por meio do Spring Data JPA (Java Persistence API), um framework de abstração de alto nível que facilita as operações de persistência e consulta de dados. Essa ferramenta automatiza a geração de implementações de repositórios com base em convenções de nomenclatura de métodos, reduzindo significativamente a necessidade de escrever código boilerplate — ou seja,

trechos repetitivos e não essenciais que seriam implementados manualmente pelo desenvolvedor.

Para atender ao requisito crítico de desempenho estabelecido (tempo de resposta ≤ 500 ms para 95% das requisições), o sistema incorpora Redis como solução de cache distribuído, permitindo armazenamento em memória de dados frequentemente acessados e reduzindo substancialmente a latência de consultas repetitivas. A implementação de cache utiliza as anotações do Spring Cache, proporcionando mecanismo declarativo para configuração de políticas de armazenamento temporário, expiração automática e invalidação seletiva de dados.

Em conformidade com os requisitos de segurança especificados, a segurança da aplicação fundamenta-se no Spring Security, framework que implementa autenticação baseada em JSON Web Tokens (JWT), proporcionando mecanismo stateless adequado para arquiteturas distribuídas. O sistema de autorização contempla três níveis hierárquicos de permissões específicos para desenvolvimento ágil: TESTER, responsável pela identificação e registro de defeitos; DEVELOPER, com permissões para resolução de bugs e desenvolvimento de funcionalidades; e SCRUM_MASTER, com acesso completo para gestão de sprints e coordenação de retrospectivas.

O controle de concorrência é gerenciado através de optimistic locking implementado via JPA, garantindo integridade transacional em operações simultâneas sem comprometer a performance do sistema. Esta abordagem é particularmente adequada para cenários com alta frequência de consultas e baixa contenção de escrita, característicos de sistemas de rastreamento de defeitos.

Para suportar os requisitos de testabilidade e facilitar o alcance da cobertura mínima de testes ≥ 80%, a documentação automatizada da API é gerada através do OpenAPI/Swagger, proporcionando interface interativa para exploração e teste dos endpoints durante o desenvolvimento e facilitando a integração com sistemas externos. O monitoramento operacional utiliza Spring Actuator, que disponibiliza endpoints especializados para verificação de saúde dos serviços e coleta de métricas de desempenho essenciais para ambientes de produção.

O sistema de logging estruturado emprega SLF4J para captura organizada de eventos de aplicação, facilitando a rastreabilidade de operações e a identificação proativa de problemas. A estruturação dos logs permite integração eficiente com ferramentas de análise e monitoramento centralizadas, essenciais para manutenção de sistemas enterprise.

4.1.2 - CAMADA DE FRONTEND

A camada de frontend foi construída utilizando React versão 18 como biblioteca principal para construção da interface de usuário, combinada com TypeScript para tipagem estática e Vite como ferramenta de build e desenvolvimento. Esta combinação tecnológica proporciona experiência de desenvolvimento otimizada, maior confiabilidade do código através da verificação de tipos em tempo de compilação e tempos de inicialização significativamente reduzidos, contribuindo para o atendimento dos requisitos de testabilidade e manutenibilidade do sistema.

A escolha do React 18 fundamenta-se em sua arquitetura baseada em componentes e nas funcionalidades avançadas de renderização concorrente, que melhoram substancialmente a responsividade da interface de usuário, alinhando-se ao requisito de tempo de resposta estabelecido. A programação declarativa característica do React permite desenvolvimento mais intuitivo e previsível, onde o estado da interface reflete diretamente o estado dos dados da aplicação através de um modelo de programação reativo.

Para atender aos requisitos de usabilidade e interface responsiva com padrões de acessibilidade, o sistema de estilização utiliza Tailwind CSS como framework de estilos utilitários, proporcionando desenvolvimento rápido e consistente da interface visual através de classes predefinidas. Esta abordagem elimina a necessidade de criação manual de folhas de estilo personalizadas e garante design system coeso em toda a aplicação. A biblioteca shadcn/ui complementa o Tailwind CSS fornecendo componentes pré-construídos e acessíveis que seguem as melhores práticas de design de interfaces modernas.

Estes componentes implementam padrões de acessibilidade estabelecidos pelo Web Content Accessibility Guidelines (WCAG), atendendo diretamente aos requisitos de usabilidade especificados e proporcionam base sólida para construção de interfaces profissionais.

O gerenciamento de estado servidor-cliente é implementado através do React Query, biblioteca especializada em sincronização de dados entre frontend e backend, contribuindo significativamente para o atendimento dos requisitos de desempenho. Esta tecnologia oferece funcionalidades avançadas como cache inteligente, refetch automático, optimistic updates e gerenciamento de estados de carregamento e erro, resultando em experiência de usuário superior e redução significativa de requisições desnecessárias ao servidor.

O roteamento da aplicação utiliza Wouter, biblioteca minimalista que proporciona navegação client-side eficiente e declarativa. Esta escolha privilegia simplicidade e performance, oferecendo funcionalidades essenciais de roteamento sem a complexidade de soluções mais robustas, adequando-se perfeitamente ao escopo e requisitos do sistema.

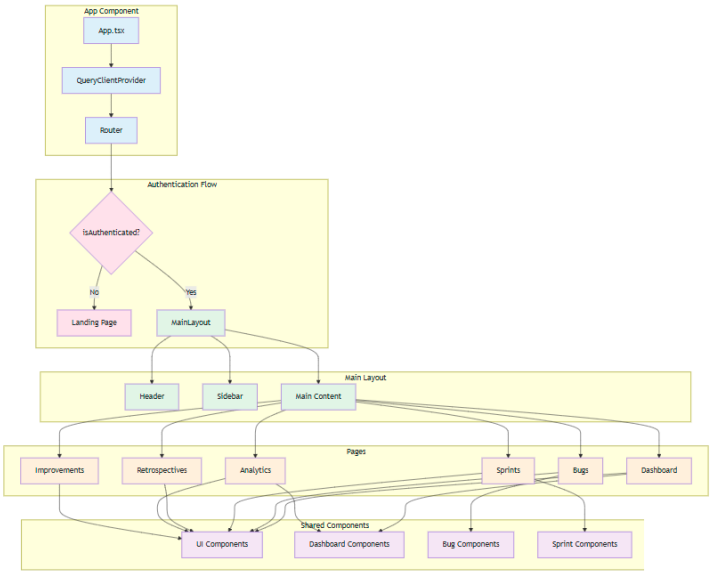
A apresentação de métricas e relatórios analíticos é implementada através da biblioteca Recharts, que oferece componentes especializados em visualização de dados construídos especificamente para React. Esta solução proporciona gráficos interativos e responsivos essenciais para apresentação das métricas de qualidade utilizadas durante as retrospectivas ágeis, incluindo densidade de bugs, cobertura de testes e análises de tendências temporais.

A arquitetura frontend baseia-se em componentes reutilizáveis que seguem princípios de composição e encapsulamento, facilitando o alcance dos requisitos de testabilidade estabelecidos. Os componentes são organizados hierarquicamente, com componentes de baixo nível responsáveis por funcionalidades específicas e componentes de alto nível que orquestram a integração entre diferentes módulos funcionais. Esta estrutura promove reutilização de código, facilita testes unitários e componentes isolados,

contribuindo para o atendimento da cobertura mínima de testes ≥ 80%, e permite evolução incremental da interface de usuário sem comprometer a estabilidade de funcionalidades existentes.

A Figura 3 apresente um esquema da arquitetura do Frontend, focando nos módulos e componentes das páginas.

Figura 3 - Diagrama da Arquitetura do Frontend



Fonte: Os autores

4.2 - ARQUITETURA GERAL

4.2.1 - CARACTERÍSTICAS ARQUITETURAIS

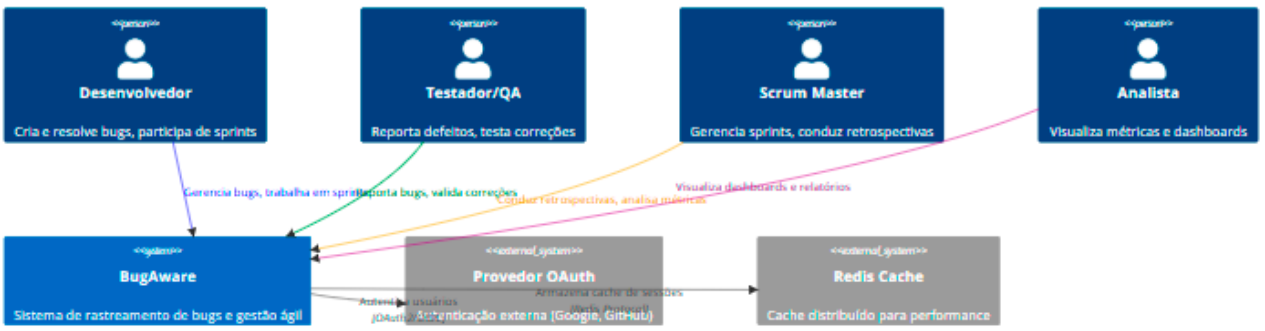
O sistema adota arquitetura orientada a serviços baseada no padrão REST, proporcionando separação clara entre as camadas de apresentação, lógica de negócio e persistência de dados. Esta estrutura modular facilita a manutenibilidade, escalabilidade e permite evolução independente dos componentes, características fundamentais para sistemas enterprise.

A API REST implementada facilita integrações futuras com sistemas externos e contribui significativamente para a manutenibilidade do sistema

através de interfaces padronizadas e bem documentadas. Paralelamente, a utilização de TypeScript na camada de apresentação reduz substancialmente a ocorrência de erros em tempo de execução, proporcionando maior robustez e confiabilidade ao sistema como um todo.

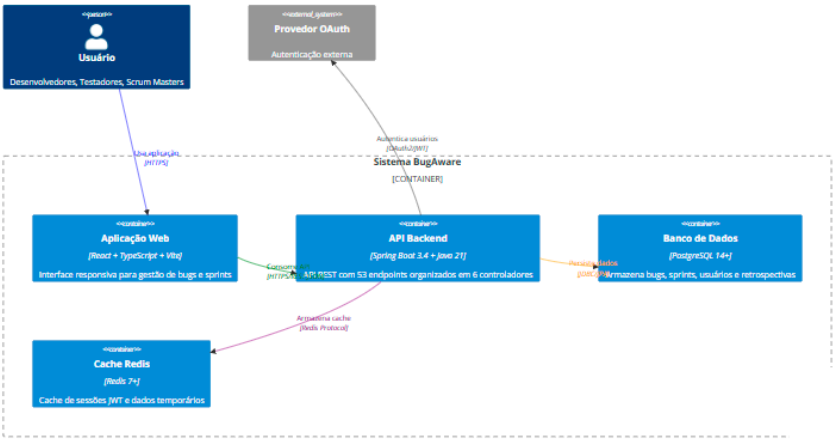
O diagrama de Contexto (Figura 4) situa o BugAwareRetro no ecossistema ágil, mostrando apenas os principais atores externos e o fluxo de informação entre eles e o sistema. A Figura 5, no nível de Container, divide a solução em frontend React/TypeScript e backend Spring Boot, indicando também serviços de suporte, como banco de dados e autenticação. No nível de Componente do backend (Figura 6) detalham-se módulos internos da API, enquanto o diagrama de Componente do frontend (Figura 7) expõe a organização de páginas e componentes React que consomem essa API.

Figura 4 - Diagrama de Contexto (Modelo C4)



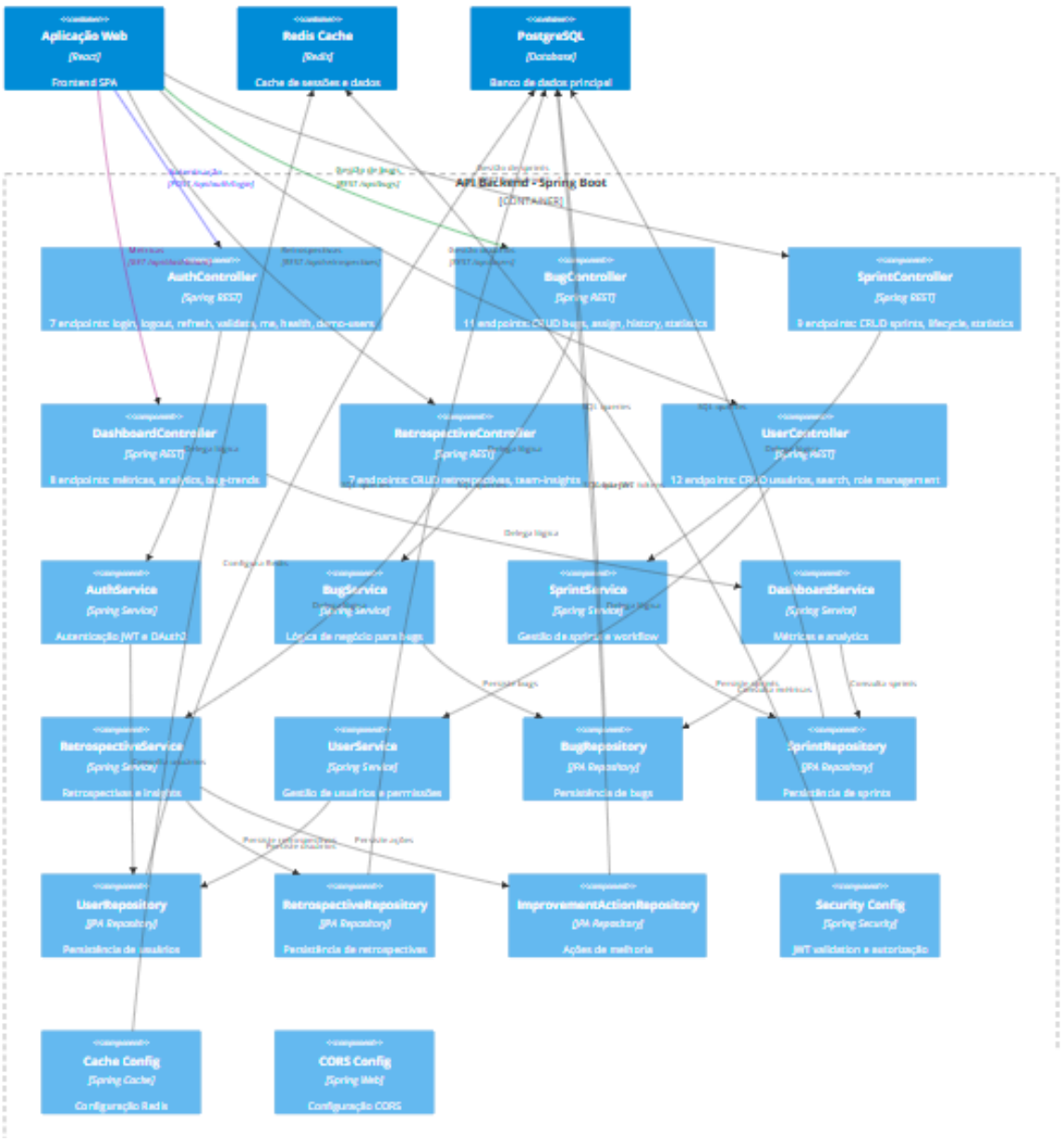
Fonte: Os autores

Figura 5 - Diagrama de Container (Modelo C4)



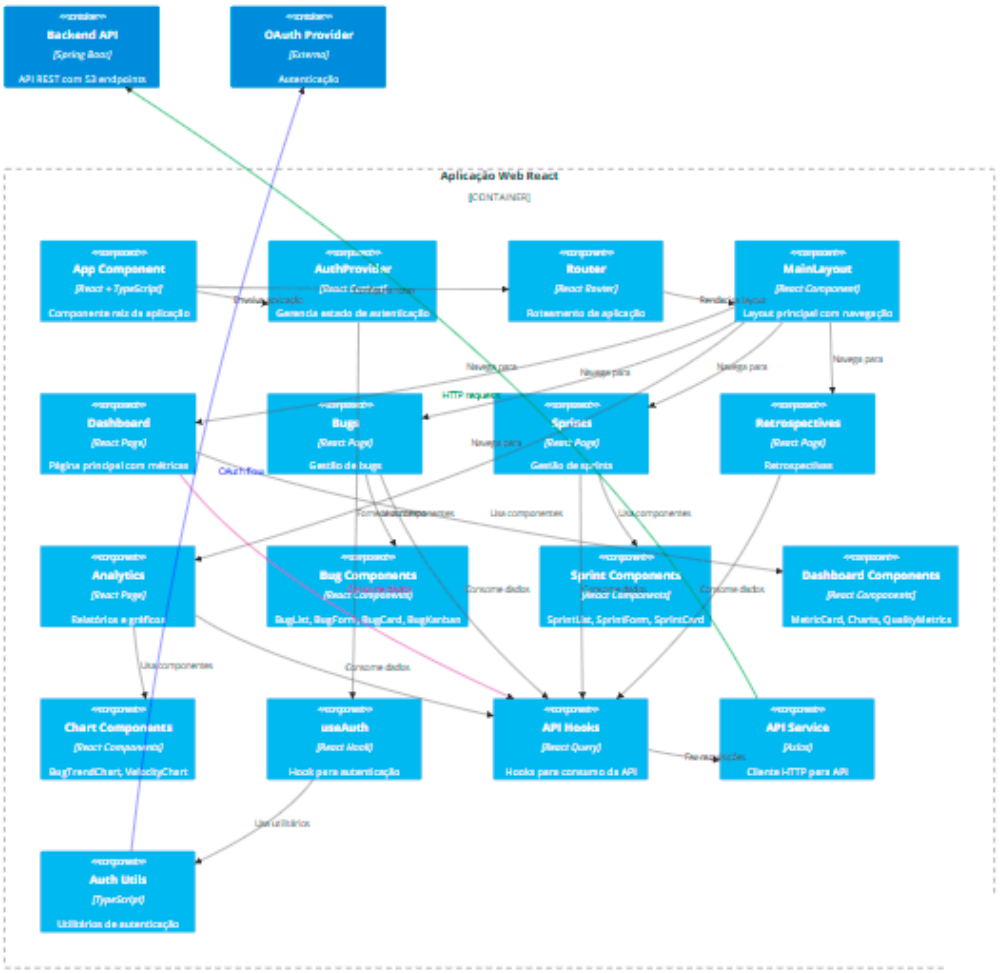
Fonte: Os autores

Figura 6 - Diagrama de Components do Backend API (Modelo C4)



Fonte: Os autores

Figura 7 - Diagrama de Components do Frontend (Modelo C4)



Fonte: Os autores

4.2.2 - PADRÕES ARQUITETURAIS IMPLEMENTADOS

A arquitetura do sistema fundamenta-se na implementação de padrões consolidados da engenharia de software. A estrutura em camadas (Layered Architecture) estabelece separação clara entre controladores, serviços e repositórios, garantindo responsabilidades bem definidas e baixo acoplamento entre os componentes.

O padrão Repository foi adotado para abstrair a camada de acesso a dados, proporcionando interface consistente para operações de persistência

independentemente da tecnologia de banco de dados subjacente. Esta implementação segue os princípios da Arquitetura Hexagonal (Ports and Adapters), onde adaptadores específicos conectam o domínio da aplicação com tecnologias externas. A separação entre interfaces JPA específicas e adaptadores de repositório garante que mudanças na tecnologia de persistência não afetem a lógica de negócio central.

A transferência de dados entre camadas utiliza o padrão Data Transfer Object (DTO), garantindo que apenas as informações necessárias sejam transmitidas e proporcionando controle granular sobre a serialização de dados. Este padrão contribui para a segurança do sistema ao evitar exposição inadvertida de dados sensíveis e otimiza a comunicação entre frontend e backend através da redução do volume de dados trafegados.

A arquitetura de frontend baseia-se em componentes reutilizáveis (Component-Based Architecture) implementados em React, promovendo modularidade e facilitando a manutenção e evolução da interface de usuário. Esta abordagem alinha-se aos princípios da engenharia de software baseada em componentes (Component-Based Software Engineering).

4.2.3 - CONFIGURAÇÕES DE SEGURANÇA E PERFORMANCE

O sistema implementa autenticação stateless através de JSON Web Tokens (JWT), eliminando a necessidade de manutenção de sessões no servidor e proporcionando escalabilidade horizontal. A autorização baseia-se em três níveis hierárquicos de permissões: TESTER, DEVELOPER e SCRUM_MASTER, cada qual com responsabilidades específicas alinhadas às práticas de desenvolvimento ágil.

A integração com provedores externos de autenticação é viabilizada através do protocolo OAuth2, oferecendo flexibilidade para organizações que utilizam sistemas de autenticação centralizados. O Spring Security proporciona configuração robusta de segurança, implementando práticas consolidadas de

proteção contra vulnerabilidades comuns em aplicações web, incluindo proteção contra ataques de injeção, cross-site scripting (XSS) e cross-site request forgery (CSRF).

Para otimização de performance, o sistema emprega Redis como solução de cache distribuído, reduzindo a latência de consultas frequentes e diminuindo a carga sobre o banco de dados principal. As anotações do Spring Cache facilitam a implementação de estratégias de cache declarativas, permitindo configuração granular de políticas de expiração e invalidação baseadas em eventos específicos da aplicação.

O controle de concorrência utiliza optimistic locking através do JPA, garantindo integridade transacional em operações simultâneas sem comprometer a performance do sistema. Esta abordagem é particularmente adequada para aplicações com alta frequência de leitura e baixa contenção de escrita, cenário típico de sistemas de rastreamento de defeitos onde consultas são mais frequentes que atualizações.

4.2.4 - DOCUMENTAÇÃO E MONITORAMENTO

A documentação da API é gerada automaticamente através do OpenAPI/Swagger, proporcionando interface interativa para exploração e teste dos endpoints durante o desenvolvimento. Esta abordagem garante que a documentação permaneça sempre atualizada e sincronizada com a implementação, eliminando discrepâncias comuns entre código e documentação em projetos de software.

O monitoramento operacional é implementado via Spring Actuator, que disponibiliza endpoints especializados para verificação de performance e estado dos serviços (health checks) e coleta de métricas de sistema. Estas funcionalidades são essenciais para implementação de práticas DevOps e monitoramento proativo da aplicação em ambiente de produção, permitindo identificação precoce de problemas e otimização contínua da performance.

O sistema de logging estruturado utiliza SLF4J para captura organizada de eventos de aplicação, facilitando a rastreabilidade de operações e a identificação de problemas em ambiente de produção. A estruturação dos logs permite integração eficiente com ferramentas de análise e monitoramento centralizadas, suportando práticas de observabilidade em sistemas distribuídos.

4.3 - PADRÕES DE PROJETO E BOAS PRÁTICAS UTILIZADAS

O desenvolvimento do Sistema BugAwareRetro utilizou padrões de projeto consolidados e práticas amplamente reconhecidas da engenharia de software, com o objetivo de assegurar a manutenibilidade, extensibilidade e robustez da solução implementada.

A implementação de Value Objects constitui elemento fundamental da arquitetura, seguindo princípios do Domain-Driven Design (DDD). Estes objetos representam conceitos do domínio através de seus valores, caracterizando-se por imutabilidade, ausência de identidade própria, auto-validação e substituibilidade. No contexto do sistema, value objects como Status, Prioridade e Período encapsulam regras de negócio específicas e garantem consistência semântica em todo o sistema.

A separação entre repositórios JPA específicos e adaptadores de repositório implementa os princípios da Arquitetura Hexagonal (Ports and Adapters) e Clean Architecture. Esta estrutura estabelece clara separação de responsabilidades onde repositórios JPA definem operações específicas do Spring Data, mantendo forte acoplamento com a infraestrutura de persistência, enquanto adaptadores implementam interfaces do domínio, conectando efetivamente a lógica de negócio com a camada de infraestrutura.

As métricas de qualidade implementadas no sistema constituem elemento central para apoio às retrospectivas ágeis, fornecendo dados objetivos essenciais para avaliação de desempenho e identificação de oportunidades de melhoria

contínua. A densidade de bugs, calculada como quantidade de defeitos por Story Point entregue, permite avaliação da qualidade do código produzido e eficácia dos processos de desenvolvimento. Esta métrica facilita discussões sobre padrões de qualidade, eficácia das práticas de desenvolvimento e estabelecimento de metas de redução de defeitos.

A métrica de acerto na primeira tentativa indica a porcentagem de funcionalidades implementadas corretamente sem necessidade de retrabalho, constituindo indicador direto da qualidade do processo e compreensão dos requisitos. A cobertura de testes representa a porcentagem do código exercitada pelos testes automatizados, sendo fundamental para avaliar confiabilidade do software e capacidade de detecção precoce de defeitos. O tempo de ciclo mede o intervalo entre início do desenvolvimento e entrega completa, proporcionando indicador crítico da eficiência do fluxo de trabalho.

4.4 - INTERFACE DO USUÁRIO

A interface do usuário foi desenvolvida seguindo princípios de design centrado no usuário e práticas de experiência de usuário (User Experience) específicas para ferramentas de desenvolvimento ágil. A arquitetura visual baseia-se em componentes reutilizáveis que garantem consistência visual e comportamental em toda a aplicação.

O design system implementado através da combinação Tailwind CSS e shadcn/ui proporciona linguagem visual coesa, com paleta de cores específica para diferentes tipos de informação, tipografia hierárquica que facilita a leitura e compreensão, e componentes interativos que fornecem feedback visual adequado para ações do usuário.

A organização da informação segue padrões estabelecidos para dashboards analíticos, com hierarquia visual clara que prioriza métricas críticas, agrupamento lógico de funcionalidades relacionadas e navegação intuitiva que reflete o fluxo de trabalho típico de equipes ágeis. A responsividade da interface garante

funcionamento adequado em diferentes dispositivos e resoluções, considerando o uso da ferramenta tanto em ambiente de escritório quanto em sessões remotas de retrospectiva.

As figuras a seguir apresentam algumas das telas resultantes da aplicação.

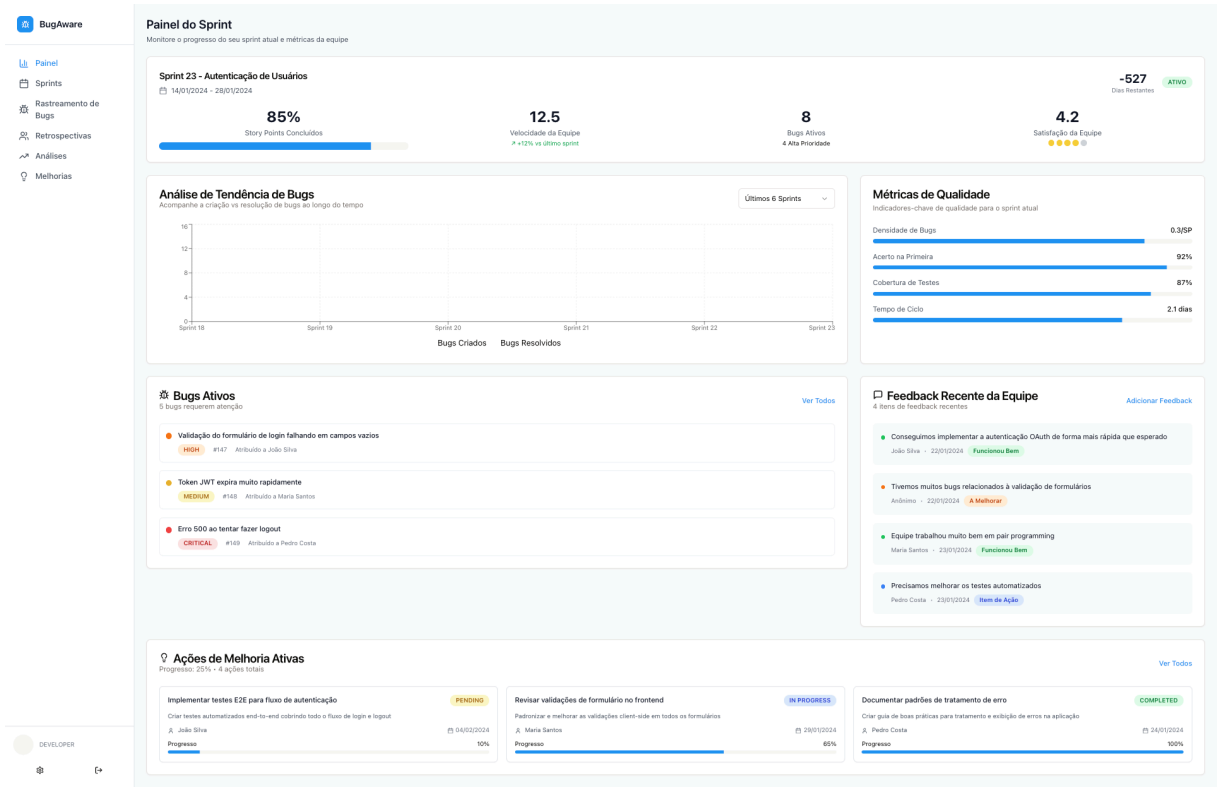
A Figura 8 apresenta a tela introdutória da aplicação BugAwareRetro. Nela, são destacados os três pilares centrais da ferramenta: insights baseados em dados, colaboração da equipe e melhoria contínua. Essa tela orienta novos usuários sobre os objetivos da plataforma de forma clara e objetiva.

Figura 8 - Tela introdutória com apresentação dos pilares

Fonte: Os autores

Na sequência, a Figura 9 apresenta o painel de resultado da sprint atual. Este painel oferece uma visão consolidada da sprint em andamento, exibindo progresso percentual, métricas de qualidade, bugs ativos em diferentes níveis de criticidade, feedbacks recentes da equipe e ações de melhoria em andamento.

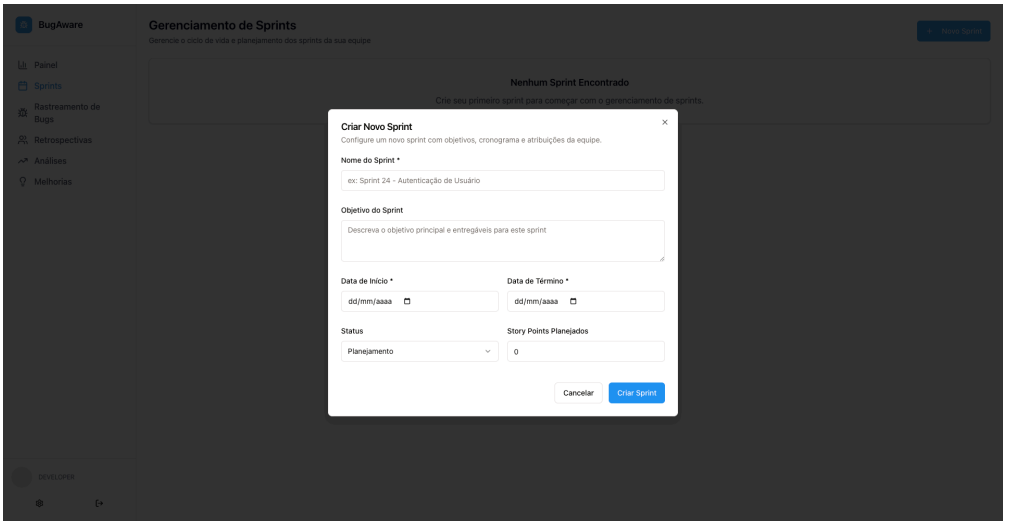
Figura 9 - Painel de resultados da sprint atual



Fonte: Os autores

A Figura 10 apresenta o formulário de criação de nova sprint. Essa tela permite que o usuário insira dados relevantes como nome, objetivo, datas de início e término, e a quantidade planejada de story points. A interface foi projetada para garantir clareza e usabilidade no planejamento de sprints, seguindo diretrizes de design centrado no usuário.

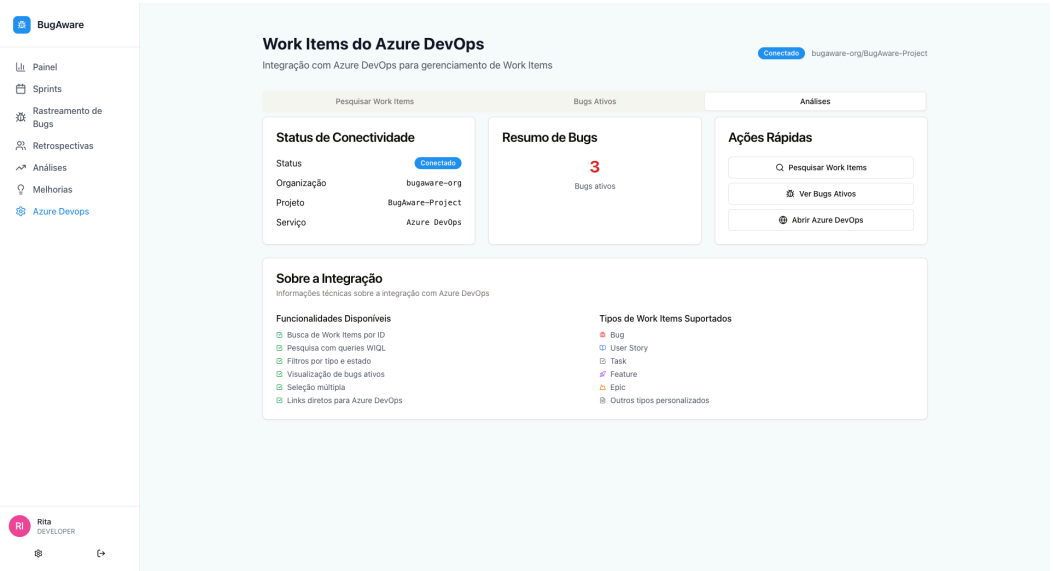
Figura 10 - Formulário de criação de nova sprint



Fonte: Os autores

Em seguida, a Figura 11 exibe o status de conectividade e as funcionalidades disponíveis referentes à integração com o Azure DevOps. O painel fornece uma visão consolidada dos bugs ativos sincronizados da ferramenta externa, permitindo ações rápidas como redirecionamento para o item original.

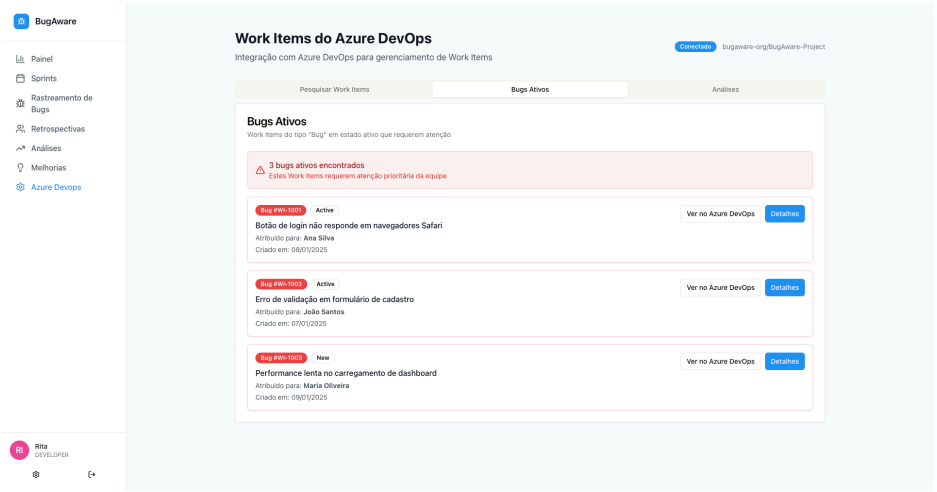
Figura 11 - Tela de integração com Azure DevOps



Fonte: Os autores

A Figura 12 apresenta a visualização de bugs ativos recuperados da plataforma Azure DevOps. São exibidos itens com status “Active” ou “New”, além de informações detalhadas como título do bug, responsável, data de criação e links para visualização externa. A sinalização em vermelho indica que os bugs requerem atenção prioritária da equipe.

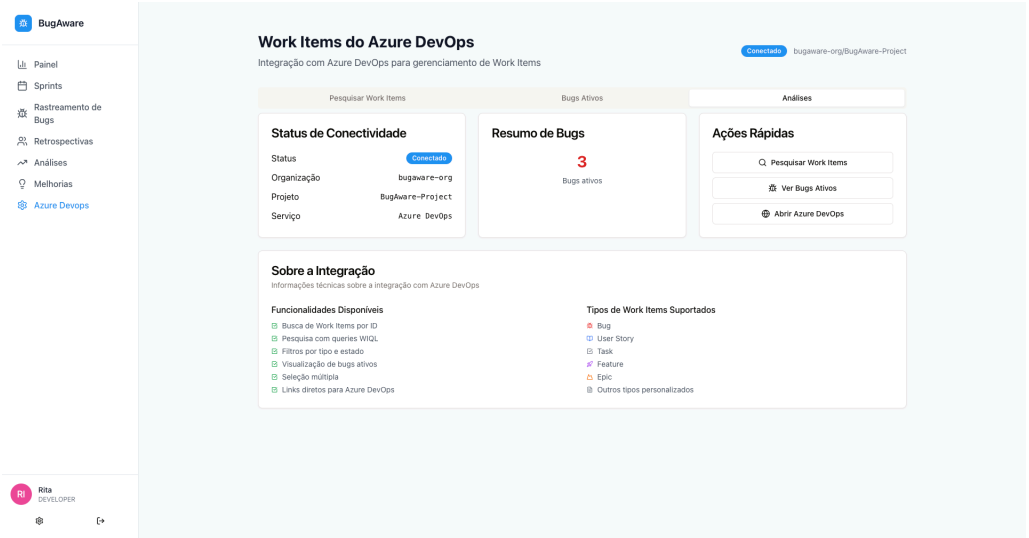
Figura 12 - Listagem de bugs ativos integrados ao Azure DevOps



Fonte: Os autores

A Figura 13 exibe a análise de Work items provenientes da Azure DevOps. Nela é possível observar a conectividade, a quantidade de problemas existentes, detalhes de integrações e ações rápidas referentes à integração.

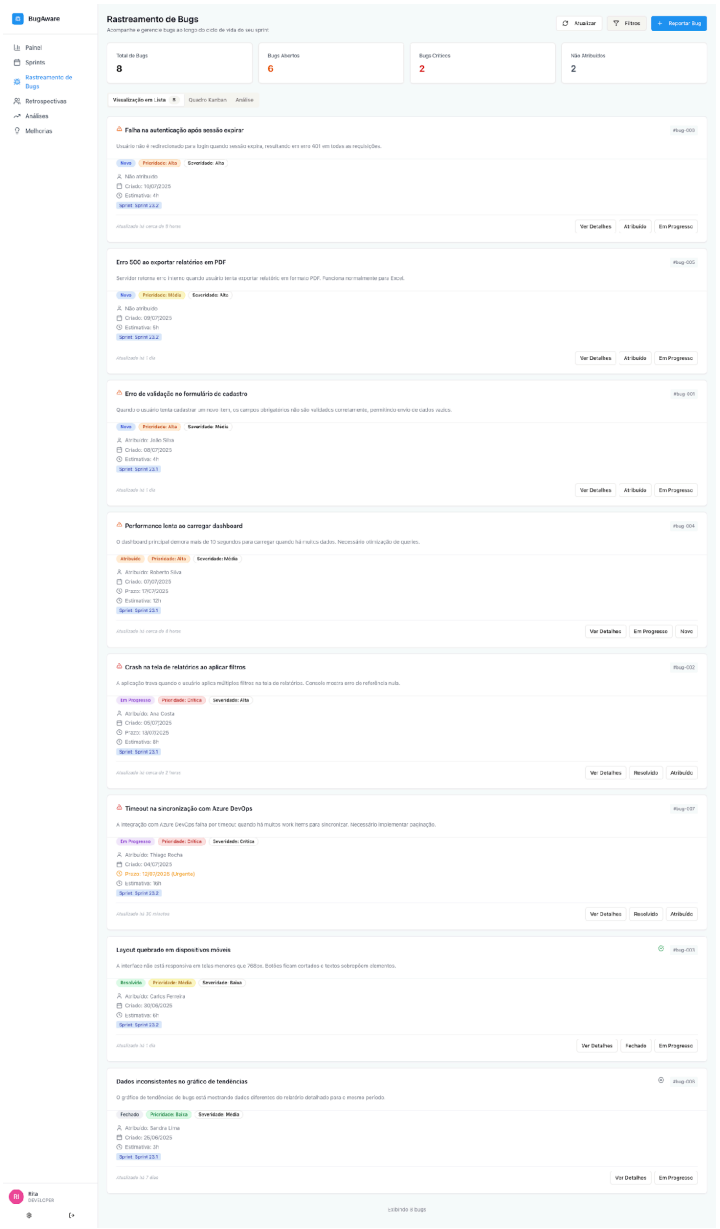
Figura 13 - Análise de Work Items da Azure DevOps



Fonte: Os autores

A Figura 14 exibe a tela de Acompanhamento de Bugs. Por meio desse painel será possível realizar a gestão completa do ciclo de vida dos bugs, desde sua criação até sua resolução.

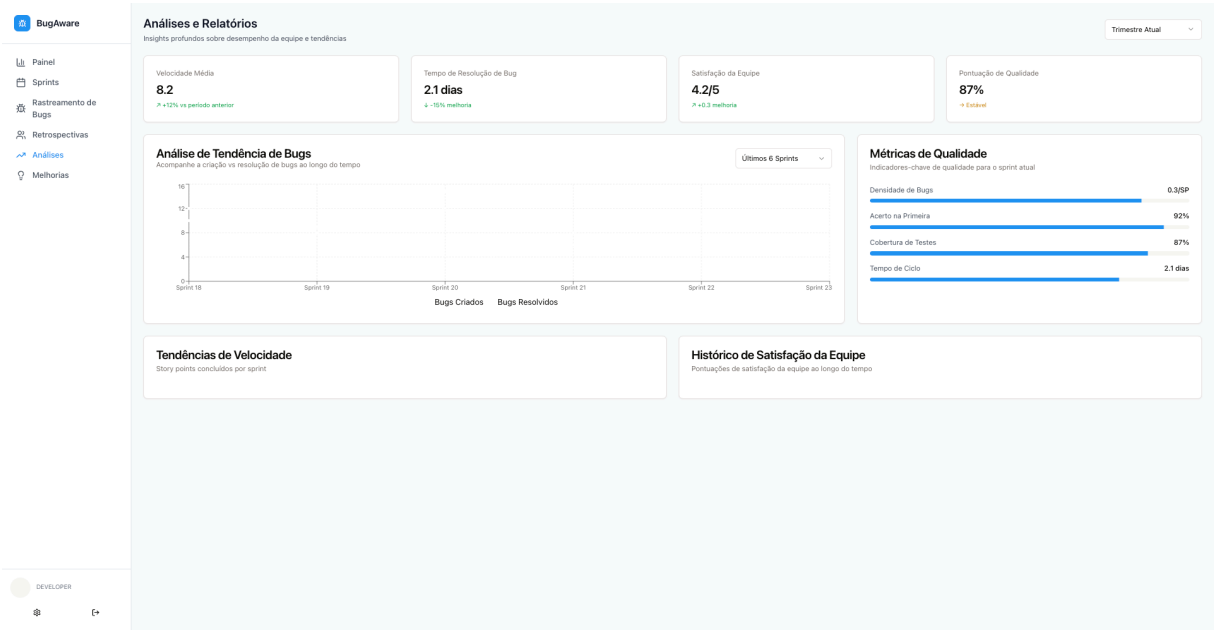
Figura 14 – Tela de Rastreabilidade de Bugs



Fonte: Os autores

A Figura 15 exibe o painel de análises gerais, com métricas de desempenho como velocidade média, tempo de resolução de bugs, satisfação da equipe e pontuação de qualidade. Além disso, são apresentadas visualizações de tendência de bugs, tendência de velocidade e histórico de satisfação da equipe ao longo do tempo.

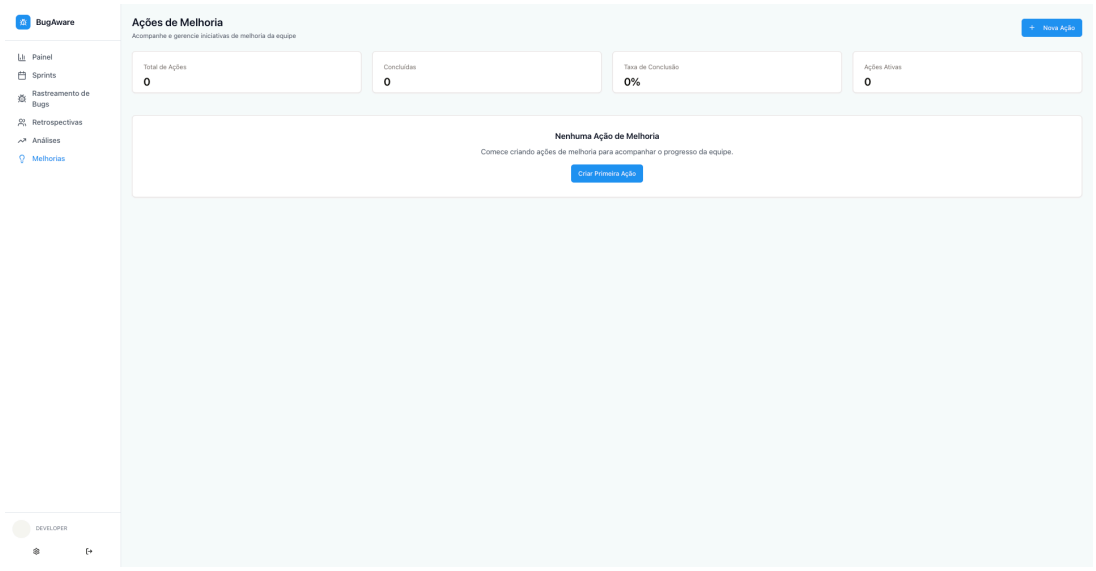
Figura 15 - Painel de análises e relatórios da equipe



Fonte: Os autores

A Figura 16 exibe a tela dedicada às ações de melhoria. Neste exemplo, observa-se a ausência de ações cadastradas, evidenciada pela interface limpa e pelo botão de criação destacado. Essa tela serve como ponto inicial para acompanhamento de planos de ação definidos durante as retrospectivas.

Figura 16 - Tela de ações de melhoria



Fonte: Os autores

4.5 - TESTES AUTOMATIZADOS

A estratégia de testes implementada no Sistema BugAwareRetro contempla múltiplas camadas de validação distribuídas entre as camadas de frontend e backend, garantindo qualidade e confiabilidade através de 82 testes automatizados. A suíte de testes atual apresenta taxa de sucesso geral de 91,5%, com 75 testes aprovados e 7 testes com falhas identificadas que requerem correção.

4.5.1. TESTES DE FRONTEND

A camada de frontend implementa 28 testes automatizados utilizando Vitest combinado com React Testing Library, alcançando taxa de sucesso de 96,4% com 27 testes aprovados. A configuração de testes utiliza ambiente jsdom para simulação do Document Object Model em ambiente Node.js, permitindo execução eficiente de testes de componentes React sem necessidade de navegador real.

Os testes de componentes concentram-se em três arquivos principais que validam funcionalidades críticas da interface de usuário. O componente BugCard é submetido a oito testes específicos que verificam renderização correta de informações de defeitos, incluindo validação de identificadores truncados, exibição condicional de desenvolvedores atribuídos, formatação de horas estimadas e aplicação adequada de cores baseadas em prioridade e status. Adicionalmente, os testes validam interatividade através da verificação de elementos clicáveis e navegação adequada.

O único teste com falha identificado relaciona-se à formatação de datas no componente BugCard, onde existe inconsistência entre o formato brasileiro esperado e o formato americano retornado pelo sistema. Esta falha, embora não crítica para funcionalidade geral, demonstra necessidade de padronização de localização para garantir experiência consistente para usuários brasileiros.

Os testes de componentes de interface de usuário incluem validação abrangente do componente Button, que implementa funcionalidades básicas de interação, e testes de funções utilitárias que garantem comportamento correto de operações auxiliares utilizadas em toda a aplicação. Esta cobertura estabelece base sólida para validação de componentes reutilizáveis que constituem a fundação da arquitetura de interface.

4.5.2 - TESTES NO BACKEND COM JUNIT

A camada de backend implementa 54 testes distribuídos em seis arquivos especializados, utilizando JUnit 5 combinados com Spring Boot Test e Testcontainers para testes de integração. A configuração emprega banco de dados H2 em memória para isolamento de testes e execução eficiente, alcançando taxa de sucesso de 88,9% com 48 testes aprovados.

Os testes de controladores validam funcionalidades essenciais da API REST, incluindo três testes do DashboardController que verificam a recuperação correta de dados analíticos para sprints específicas e ativas. Estes testes garantem

que métricas essenciais para retrospectivas sejam calculadas e apresentadas adequadamente, validando integração entre camadas de serviço e apresentação.

Os testes de integração implementados através de BugServiceIntegrationTest validam comunicação entre diferentes camadas da aplicação, garantindo que operações complexas de criação e manipulação de defeitos funcionem corretamente em cenário próximo ao ambiente de produção. Estes testes utilizam Testcontainers para provisionar a infraestrutura de teste isolada, incluindo banco de dados e serviços auxiliares.

A validação de regras de negócio é implementada através de testes especializados para BugService, que verificam conformidade com requisitos funcionais específicos relacionados ao ciclo de vida de defeitos. Testes de workflow de status garantem que transições de estado de defeitos sigam regras estabelecidas e não permitam mudanças inválidas que comprometam a integridade dos dados.

As seis falhas identificadas concentram-se principalmente em problemas de inicialização de dados do banco de dados e configuração de contexto de teste, manifestando-se através de NullPointerException em operações de criação de defeitos. Estas falhas indicam necessidade de revisão na configuração de dependências em ambiente de teste, particularmente relacionadas à injeção de dependências e inicialização de beans do Spring.

5 - CONSIDERAÇÕES FINAIS

5.1 - CONCLUSÃO

Apesar das limitações de tempo impostas ao desenvolvimento do sistema BugAwareRetro, a equipe conseguiu alcançar os objetivos propostos, ao conceber e implementar uma ferramenta capaz de apoiar de forma efetiva as reuniões de retrospectiva. O sistema contribuiu para a valorização desse momento essencial da metodologia ágil, frequentemente negligenciado, ao proporcionar maior clareza na identificação de falhas e no acompanhamento de ações corretivas. A aplicação foi amplamente documentada, contemplando aspectos arquiteturais, funcionais e técnicos, o que favorece sua manutenção e evolução. No entanto, reconhece-se que ainda há espaço para aprimoramentos, especialmente no que tange à escalabilidade, usabilidade e integração com outras ferramentas de gestão de projetos e qualidade de software.

5.2 - LIMITAÇÕES DO TRABALHO

Em função das restrições de tempo enfrentadas durante o desenvolvimento do sistema BugAwareRetro, nem todos os requisitos inicialmente planejados puderam ser plenamente implementados. Funcionalidades como a integração com múltiplas ferramentas de rastreamento de bugs, a autenticação por meio de Single Sign-On (SSO) e o suporte multiplataforma não foram contempladas nesta versão. Além disso, recursos previstos como a exportação de relatórios analíticos e o gerenciamento avançado de responsabilidades por papéis (controle de permissões e hierarquia de usuários) permaneceram fora do escopo entregue. Tais limitações, embora reconhecidas, não comprometeram o objetivo principal do projeto, que era disponibilizar uma ferramenta funcional e eficaz para apoiar reuniões de retrospectiva ágil com foco na rastreabilidade de falhas e melhoria contínua.

5.3 - LIÇÕES APRENDIDAS

Ao longo da graduação em Engenharia de Software, observou-se uma estreita relação entre a formação acadêmica e a prática profissional exercida

paralelamente. A atuação como bolsista de desenvolvimento web por sete anos na Universidade Federal do Rio Grande do Norte (UFRN) proporcionou oportunidades contínuas de aplicação prática dos conteúdos adquiridos em disciplinas fundamentais, tais como processos de software, levantamento e análise de requisitos, modelagem de sistemas e desenvolvimento web. Destaca-se também que, nesse período, a experiência com administração de servidores Linux foi um diferencial relevante, contribuindo para uma visão mais abrangente da infraestrutura necessária ao funcionamento de aplicações robustas.

Nos últimos dois anos, a atuação como desenvolvedora web e engenheira de software em empresas de segmentos distintos consolidou ainda mais os conhecimentos adquiridos. Durante esse período, foi possível contribuir tecnicamente com as equipes de desenvolvimento em três frentes essenciais: qualidade de software, conformidade com requisitos funcionais e não funcionais, e clareza técnica nas decisões de projeto. Essa atuação envolveu atividades como apoio à definição de padrões de codificação, avaliação de aderência aos critérios de aceitação, revisão de arquitetura de soluções e facilitação da comunicação técnica entre os membros da equipe. Tais responsabilidades são frequentemente atribuídas a profissionais com perfil técnico sênior ou com experiência em liderança técnica, ainda que não formalmente alocados nessa função.

Conclui-se, portanto, que os conhecimentos desenvolvidos ao longo do curso foram significativamente aprimorados por meio da prática profissional constante, resultando em um ciclo contínuo de aprendizado e aplicação, essencial para a consolidação das competências exigidas na engenharia de software contemporânea.

5.4 - TRABALHOS FUTUROS

Para consolidar e expandir as capacidades do Sistema BugAwareRetro como ferramenta de rastreabilidade de bugs e gestão de retrospectivas, identificam-se oportunidades significativas de aprimoramento que podem ser abordadas em pesquisas e desenvolvimentos futuros. Essas melhorias visam não

apenas ampliar a funcionalidade do sistema, mas também otimizar sua eficiência operacional e capacidade analítica.

5.4.1 - INTEGRAÇÃO COM SPRING BATCH PARA PROCESSAMENTO DE DADOS EM LOTE

A primeira linha de investigação proposta consiste na implementação de funcionalidades de processamento de dados em lote através do framework Spring Batch. Esta abordagem permitiria ao sistema processar grandes volumes de dados de forma eficiente e confiável, expandindo significativamente suas capacidades de ingestão de informações.

A integração com Spring Batch possibilitara a leitura e processamento automatizado de arquivos CSV contendo registros históricos de bugs, métricas de desempenho de equipes e feedbacks de retrospectivas provenientes de sistemas externos. Esta funcionalidade seria particularmente valiosa para organizações que necessitam migrar dados históricos de ferramentas legadas ou consolidar informações dispersas em múltiplas plataformas.

Além da capacidade de importação, o Spring Batch ofereceria mecanismos robustos para validação, transformação e limpeza de dados, garantindo a integridade das informações processadas. A implementação de checkpoints e mecanismos de recuperação de falhas asseguraria a confiabilidade do processo de migração, mesmo em cenários de grandes volumes de dados ou falhas intermitentes de sistema.

5.4.2 - GERAÇÃO AUTOMATIZADA DE RESUMOS ANALÍTICOS

A segunda proposta de trabalho futuro envolve o desenvolvimento de capacidades avançadas de geração de resumos e relatórios analíticos utilizando o Spring Batch. Esta funcionalidade permitiria a criação automatizada de artefatos de dados agregados, proporcionando insights valiosos sobre o desempenho das equipes e tendências de qualidade ao longo do tempo.

O sistema seria capaz de gerar sumários de desempenho por sprint, análises de tendências de qualidade, relatórios de produtividade de equipes e métricas comparativas entre diferentes períodos. Esses relatórios poderiam ser

executados em horários pré-definidos, reduzindo significativamente a carga computacional durante os períodos de maior utilização do sistema.

A implementação desta funcionalidade contribuiria para a transformação de dados brutos em informações estratégicas, facilitando a tomada de decisões baseadas em evidências por parte dos gestores de projeto e líderes técnicos. A geração automatizada de relatórios também reduziria a necessidade de consultas manuais frequentes, otimizando o desempenho geral do sistema.

5.4.3 - IMPLEMENTAÇÃO DE COMUNICAÇÃO EM TEMPO REAL

A terceira linha de pesquisa concentra-se na implementação de mecanismos de comunicação em tempo real através de webhooks e websockets. Esta funcionalidade representaria um avanço significativo na experiência do usuário, permitindo atualizações instantâneas da interface sem a necessidade de recarregamento manual das páginas.

A implementação de webhooks possibilitaria a criação de um sistema de notificações assíncronas, permitindo que eventos importantes, como a criação de novos bugs, mudanças de status ou conclusão de retrospectivas, sejam comunicados instantaneamente aos stakeholders relevantes. Esta abordagem melhoraria significativamente a responsividade do sistema e a colaboração entre os membros das equipes.

Paralelamente, a integração de websockets permitiria atualizações em tempo real da interface do usuário, refletindo mudanças no estado dos bugs, progresso das retrospectivas e métricas de desempenho instantaneamente para todos os usuários conectados. Essa funcionalidade seria especialmente valiosa durante sessões de retrospectiva colaborativas, ou seja, quando múltiplos usuarios necessitam visualizarem simultaneamente as mesmas informações atualizadas.

As três linhas de pesquisa propostas complementam-se mutuamente, formando um conjunto coeso de melhorias que elevaria significativamente as capacidades do Sistema BugAwareRetro. A implementação dessas funcionalidades não apenas aprimoraria a experiência do usuário, mas também

posicionaria o sistema como uma ferramenta mais robusta e escalável para organizações de diferentes portes.

O desenvolvimento dessas funcionalidades futuras requer investigação adicional em aspectos como escalabilidade, segurança e integração com sistemas externos, representando oportunidades valiosas para pesquisas acadêmicas e desenvolvimento tecnológico na área de ferramentas de gestão de qualidade de software.

REFERÊNCIAS

ALMEIDA, E. S.; CARNEIRO, G. A. A. A. Análise de Métricas Ágeis para Retrospectivas. Agile Metrics Journal, v. 5, p. 1-10, 2023.

AHMAD, M. O. et al. A framework for agile process assessment using process data. Journal of Systems and Software, v. 143, p. 88–101, 2018.

ANDERSON, D. J. Kanban: Successful evolutionary change for your technology business. Blue Hole Press, 2010.

BECK, K. (2003). Test-Driven Development: By Example. Addison-Wesley Professional.

BOEHM, B. W. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

CARTAXO, B. et al. Evidence-Based Software Engineering: A systematic literature review. Information and Software Technology, v. 127, p. 106371, 2020.

CHEN, L. Continuous software engineering: A new paradigm for software development. Journal of Systems and Software, v. 118, p. 147–156, 2020.

COHN, M. User stories applied: for agile software development. Boston: Addison-Wesley, 2004. p. 67-89.

DAVIS, A. M. Software Requirements: Objects, Functions, and States. Englewood Cliffs: Prentice-Hall, 1993. p. 78-95.

DERBY, E.; LARSEN, D. Agile Retrospectives: Making Good Teams Great. Pragmatic Bookshelf, 2006. p. 23-45.

EVANS, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. p. 167-189.

ERDOĞAN, O., PEKKAYA, M. E., and GÖK, H. (2018). More effective sprint retrospective with statistical analysis. Journal of Software: Evolution and Process, 30(5).

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. ACM Transactions on Internet Technology, v. 2, n. 2, p. 115–150, 2002.

FITZGERALD, B.; STOL, K. J. Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, v. 123, p. 176–189, 2017.

FOWLER, M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002.

FOWLER, M. Command Query Responsibility Segregation (CQRS). Martin Fowler's Blog, 2011. Disponível em: https://martinfowler.com/bliki/CQRS.html. Acesso em: 04 jul. 2025.

GAME-BASED Sprint retrospectives: multiple action research. PubMed Central, 2021.

HARDT, D. The OAuth 2.0 Authorization Framework. RFC 6749, 2012. p. 1-76.

HEVNER, A. R. et al. Design science in information systems research. MIS Quarterly, v. 28, n. 1, p. 75-105, 2004.

HOWARD, M.; LEBLANC, D. Writing Secure Code. 2. ed. Redmond: Microsoft Press, 2003. p. 123-145.

HUMBLE, J.; FARLEY, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2019. p. 234.

IEEE. IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998, 1998. p. 1-40.

IMAI, M. Kaizen: The Key to Japan's Competitive Success. New York: McGraw-Hill, 1986.

INFED. Peter Senge and the learning organization. Disponível em: https://infed.org/mobi/peter-senge-and-the-learning-organization/. Acesso em: 02 jul. 2025.

JOHANNESSON, P.; PERJONS, E. An Introduction to Design Science. 2. ed. Heidelberg: Springer, 2019. p. 89-123.

JONES, M. et al. JSON Web Token (JWT). RFC 7519, 2015. p. 1-30.

KERTH, N. L. Project Retrospectives: A Handbook for Team Reviews. New York: Dorset House Publishing, 2001.

KITCHENHAM, B.; PFLEEGER, S. L. Personal opinion surveys. In: Guide to Advanced Empirical Software Engineering. Springer, 2008. p. 63-92.

KNIBERG, H.; SKARIN, M. Kanban and Scrum - Making the Most of Both. C4Media, 2010.

LEHTINEN, T. O. A. et al. A tool supporting root cause analysis for synchronous retrospectives in distributed software teams. Information and Software Technology, v. 56, n. 4, p. 408-437, 2014.

LEHTINEN, T. O. A.; ITKONEN, J.; LASSENIUS, C. Recurring opinions or productive improvements – What agile teams actually discuss in retrospectives. Empirical Software Engineering, v. 22, n. 5, 2017.

LEHTINEN, Timo O.A.; MÄNTYLÄ, Mika V.; VANHANEN, Jari; ITKONEN, Juha; LASSENIUS, Casper. Perceived Causes of Software Project Failures – An Analysis of their Relationships. Information and Software Technology, 2014.

LEI, H. et al. The application of RFID technology and Kanban management in production planning and control. Journal of Manufacturing Systems, v. 43, p. 187-195, 2017.

MARSDEN, N. et al. Psychological safety in agile retrospectives. 2021.

MATTHIES, C.; DOBRIGKEIT, F. Experience vs Data: A Case for More Data-Informed Retrospective Activities. In: Lean and Agile Software Development. Lecture Notes in Business Information Processing, v. 394, Springer, 2021.

MATTHIES, C.; HESSE, G. Towards Using Data to Inform Decisions in Agile Software Development: Views of Available Data. In: Proceedings of the 14th International Conference on Software Technologies. [S.l.]: SciTePress, 2019.

MERIDIAN UNIVERSITY. Peter Senge: The Generational Visionary of the Learning Organization. 2024. Disponível em: https://meridianuniversity.edu/content/peter-senge-the-generational-visionary-of-the-learning-organization. Acesso em: 11 jun. 2025.

MILANI, A. M. P. et al. Exploring Retrospective Meeting Practices and the Use of Data in Agile Teams. In: Proceedings of the 2025 IEEE/ACM 18th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE). [S.l.]: IEEE, 2025.

MILANI, A. M. P. et al. Retrospective Tools and Practices: A Comparative Study. arXiv preprint arXiv:2401.00000, 2024.

MINING FOR PROCESS IMPROVEMENTS: Analyzing Software Repositories in Agile Retrospectives. arXiv, 2020.

NASA. Error Cost Escalation Through the Project Life Cycle. NASA Technical Reports, 2010. Disponível em: https://ntrs.nasa.gov. Acesso em: 27 mai. 2025.

NIELSEN, J. Usability Engineering. San Francisco: Morgan Kaufmann, 1993. p. 135-156.

NOTTA. 50+ Agile Statistics You Need to Know in 2025. 2024. Disponível em: https://www.notta.ai/en/blog/agile-statistics. Acesso em: 03 jul. 2025.

OHNO, T. Toyota Production System: Beyond Large-Scale Production. New York: Productivity Press, 1988.

OSMANI, A. Learning JavaScript Design Patterns. O'Reilly Media, 2017. p. 167-189.

PEFFERS, K. et al. Design science research process: a model for producing and presenting information systems research. Journal of Management Information Systems, v. 24, n. 3, p. 1-18, 2020.

PETERSEN, K.; WOHLIN, C. Measuring and managing cycle time in agile software development. Information and Software Technology, v. 109, p. 265–275, 2019.

POPPENDIECK, M.; POPPENDIECK, T. Lean Software Development: An Agile Toolkit. IEEE Software, v. 20, n. 3, p. 102-106, 2003.

PRESSMAN, R. S.; MAXIM, B. R. Software Engineering: A Practitioner's Approach. 9. ed. New York: McGraw-Hill, 2020. p. 234-267.

ROBERTSON, S.; ROBERTSON, J. Mastering the Requirements Process: Getting Requirements Right. 3. ed. Upper Saddle River: Addison-Wesley, 2012.

RODRÍGUEZ, P., MARKKULA, J., OIVO, M., & TURULA, K. (2022). Survey on agile and lean usage in Finnish software industry. ACM Computing Surveys, 48(2), 234-256.

SCHWABER, K.; SUTHERLAND, J. The Scrum Guide. Scrum.org, 2020. Disponível em: https://scrumguides.org. Acesso em: 29 jun. 2025.

SCHWABER, K.; SUTHERLAND, J. The Scrum Guide. Scrum.org, 2017.

SCHWABER, K.; SUTHERLAND, J. The Scrum Guide. Scrum.org, 2013.

SENGE, P. M. The Fifth Discipline: The Art and Practice of the Learning Organization. New York: Doubleday, 1990.

SEOW, S. C. Designing and Engineering Time: The Psychology of Time Perception in Software. Boston: Addison-Wesley, 2008.

SINGH, J.; SINGH, H. Continuous improvement philosophy – literature review and directions. Benchmarking: An International Journal, v. 22, n. 1, p. 75-119, 2015.

SOMÉ, S. S. Supporting use case based requirements engineering. Information and Software Technology, v. 48, n. 1, p. 43-58, 2007.

SOMMERVILLE, I. Software Engineering. 10. ed. Boston: Pearson, 2015. p. 156-178.

SPICHKOVA, M. et al. Agile Retrospectives: What went well? What didn't go well? What should we do? ENASE 2025.

SUÁREZ-BARRAZA, M. F.; SMITH, T.; DAHLGAARD-PARK, S. M. Kaizen-Kata: An alternative approach to achieving continuous improvement. International Journal of Lean Six Sigma, v. 3, n. 4, p. 298–313, 2012.

SZYPERSKI, C. Component Software: Beyond Object-Oriented Programming. 2. ed. Boston: Addison-Wesley, 2002.

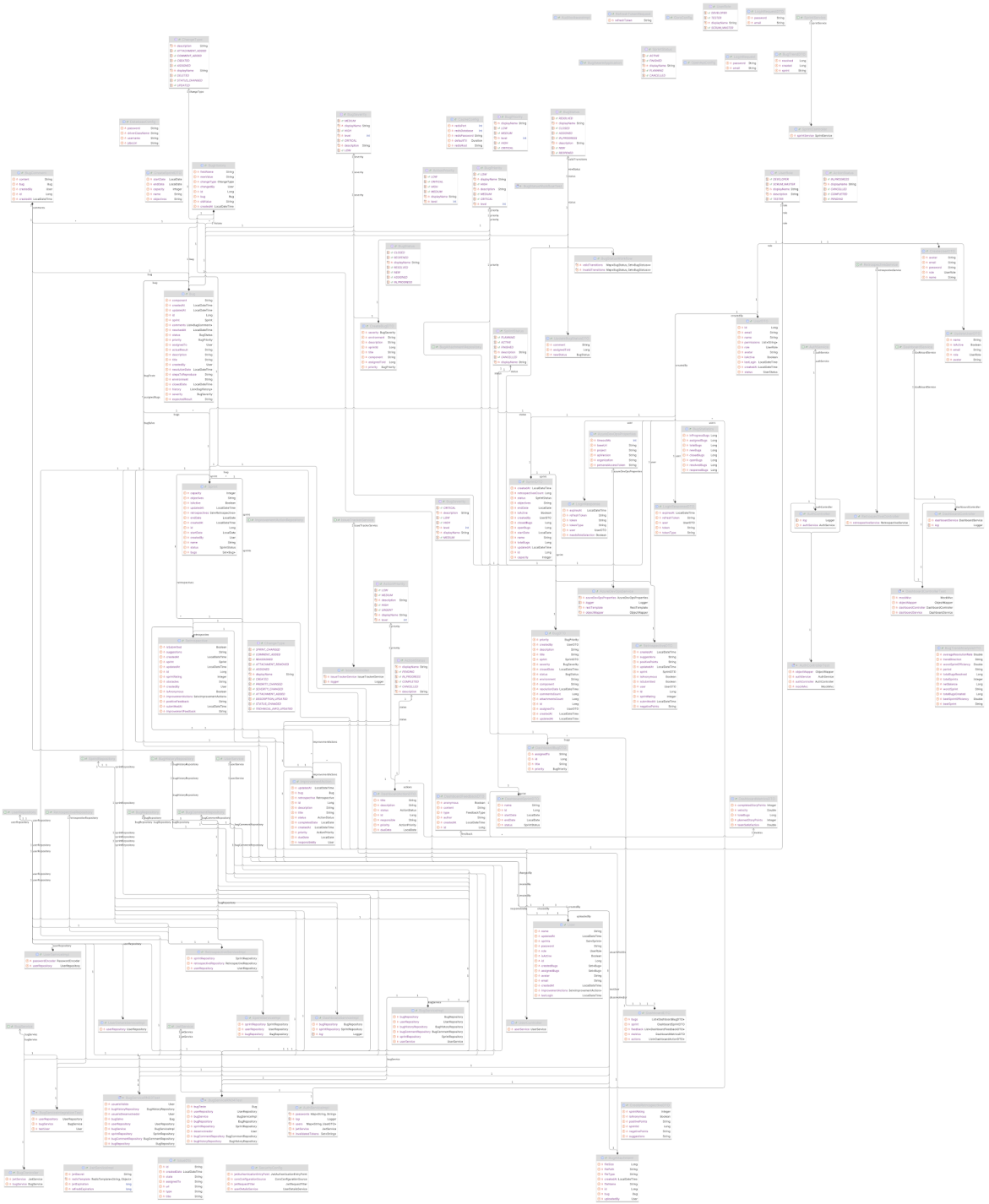
CHARALAMPIDOU, Sofia; AMPATZOGLOU, Apostolos; KAROUNTZOS, Evangelos; AVGERIOU, Paris. Empirical studies on software traceability: a mapping study. Journal of Software: Evolution and Process, [S. l.], v. 33, n. 2, p. e2294, 2021. DOI: 10.1002/smr.2294. Disponível em: https://ruomoplus.lib.uom.gr/handle/8000/598. Acesso em: 22 maio 2025.

WIEGERS, K.; BEATTY, J. Software Requirements. 3. ed. Redmond: Microsoft Press, 2013.

WIERINGA, R. J. Design Science Methodology for Information Systems and Software Engineering. Berlin: Springer, 2014.

ANEXOS

ANEXO A – DIAGRAMA DE CLASSES DO SISTEMA BUGAWARERETRO



ANEXO B – ISSUETRACKERSERVICE - INTEGRAÇÃO AZURE DEVOPS

Classe responsável pela integração com o Azure DevOps, conforme arquivo IssueTrackerService.java.

```
package com.bugaware.service;

import com.bugaware.dto.IssueDto;
import java.util.List;

/**
 * Interface genérica para serviços de rastreamento de issues
externos
 *
 * Esta interface permite a implementação de diferentes provedores
de issue tracking
 * (Azure DevOps, Jira, GitHub Issues, etc.) de forma padronizada.
 *
 * Métodos síncronos e simples para facilitar a implementação e
manutenção.
 */
public interface IssueTrackerService {

    /**
     * Busca uma issue específica pelo ID
     *
     * @param id ID da issue no sistema externo
     * @return IssueDto com dados da issue ou null se não
encontrada
     * @throws RuntimeException se ocorrer erro na comunicação com
o sistema externo
     */
    IssueDto getIssueById(String id);

    /**
     * Busca múltiplas issues pelos IDs fornecidos
     *
     * @param ids Lista de IDs das issues a serem buscadas
     * @return Lista de IssueDto com as issues encontradas (IDs não
encontrados são omitidos)
     * @throws RuntimeException se ocorrer erro na comunicação com
o sistema externo
     */
    List<IssueDto> getIssuesByIds(List<String> ids);

    /**
```

```
     * Pesquisa issues usando uma query de busca
     *
     * @param query String de busca (formato depende do provedor)
     * @return Lista de IssueDto com as issues que correspondem à
busca
     * @throws RuntimeException se ocorrer erro na comunicação com
o sistema externo
     */
    List<IssueDto> searchIssues(String query);
}
```

ANEXO C – AZUREDEVOPSSERVICE - INTEGRAÇÃO AZURE DEVOPS

Classe responsável pela integração com o Azure DevOps, conforme arquivo AzureDevOpsService.java.

```
package com.bugaware.service.impl;

import com.bugaware.config.AzureDevOpsProperties;
import com.bugaware.dto.IssueDto;
import com.bugaware.service.IssueTrackerService;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestClientException;
import org.springframework.web.client.RestTemplate;

import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.stream.Collectors;

/**
 * Implementação do serviço de integração com Azure DevOps Work
Items
 *
 * Utiliza RestTemplate para chamadas REST síncronas e autenticação
via PAT (Personal Access Token)
 * Mapeia Work Items do Azure DevOps para o DTO comum IssueDto
 */
@Service
public class AzureDevOpsService implements IssueTrackerService {

    private static final Logger logger =
LoggerFactory.getLogger(AzureDevOpsService.class);

    private final AzureDevOpsProperties azureDevOpsProperties;
    private final RestTemplate restTemplate;
    private final ObjectMapper objectMapper;
```

```
    @Autowired
    public AzureDevOpsService(AzureDevOpsProperties
azureDevOpsProperties,
                              RestTemplateBuilder
restTemplateBuilder,
                              ObjectMapper objectMapper) {
        this.azureDevOpsProperties = azureDevOpsProperties;
        this.objectMapper = objectMapper;
        this.restTemplate = restTemplateBuilder

.setConnectTimeout(Duration.ofMillis(azureDevOpsProperties.getTimeo
utMs()))

.setReadTimeout(Duration.ofMillis(azureDevOpsProperties.getTimeoutM
s()))
                .build();

        logger.info("Azure DevOps Service inicializado para
organização: {}, projeto: {}",
                    azureDevOpsProperties.getOrganization(),
azureDevOpsProperties.getProject());
    }

    @Override
    public IssueDto getIssueById(String id) {
        try {
            logger.debug("Buscando Work Item por ID: {}", id);

            String url =
String.format("%s/%s?api-version=%s&$expand=fields",

azureDevOpsProperties.getWorkItemsApiUrl(),
                                      id,

azureDevOpsProperties.getApiVersion());

            HttpHeaders headers = createAuthHeaders();
            HttpEntity<?> requestEntity = new
HttpEntity<>(headers);

            ResponseEntity<String> response =
restTemplate.exchange(
                url, HttpMethod.GET, requestEntity, String.class);

            if (response.getStatusCode() == HttpStatus.OK &&
response.getBody() != null) {
                JsonNode workItem =
objectMapper.readTree(response.getBody());
                return mapWorkItemToIssueDto(workItem);
            } else {
```

```
                logger.warn("Work Item com ID {} não encontrado ou
resposta vazia", id);
                return null;
            }

        } catch (RestClientException e) {
            logger.error("Erro ao buscar Work Item por ID {}: {}",
id, e.getMessage());
            throw new RuntimeException("Erro na comunicação com
Azure DevOps: " + e.getMessage(), e);
        } catch (Exception e) {
            logger.error("Erro inesperado ao buscar Work Item por
ID {}: {}", id, e.getMessage());
            throw new RuntimeException("Erro interno ao processar
Work Item: " + e.getMessage(), e);
        }
    }

    @Override
    public List<IssueDto> getIssuesByIds(List<String> ids) {
        if (ids == null || ids.isEmpty()) {
            logger.debug("Lista de IDs vazia, retornando lista
vazia");
            return new ArrayList<>();
        }

        try {
            logger.debug("Buscando {} Work Items por IDs: {}",
ids.size(), ids);

            String idsParam = String.join(",", ids);
            String url =
String.format("%s?ids=%s&api-version=%s&$expand=fields",

azureDevOpsProperties.getWorkItemsApiUrl(),
                                    idsParam,

azureDevOpsProperties.getApiVersion());

            HttpHeaders headers = createAuthHeaders();
            HttpEntity<?> requestEntity = new
HttpEntity<>(headers);

            ResponseEntity<String> response =
restTemplate.exchange(
                url, HttpMethod.GET, requestEntity, String.class);

            if (response.getStatusCode() == HttpStatus.OK &&
response.getBody() != null) {
                JsonNode workItemsResponse =
```

```
objectMapper.readTree(response.getBody());
                JsonNode valueNode =
workItemsResponse.get("value");

                if (valueNode != null && valueNode.isArray()) {
                    List<IssueDto> issues = new ArrayList<>();
                    for (JsonNode workItem : valueNode) {
                        IssueDto issue =
mapWorkItemToIssueDto(workItem);
                        if (issue != null) {
                            issues.add(issue);
                        }
                    }
                    logger.debug("Encontrados {} Work Items de {}
solicitados", issues.size(), ids.size());
                    return issues;
                }
            }

            logger.warn("Nenhum Work Item encontrado para os IDs
fornecidos");
            return new ArrayList<>();

        } catch (RestClientException e) {
            logger.error("Erro ao buscar Work Items por IDs {}:
{}", ids, e.getMessage());
            throw new RuntimeException("Erro na comunicação com
Azure DevOps: " + e.getMessage(), e);
        } catch (Exception e) {
            logger.error("Erro inesperado ao buscar Work Items por
IDs {}: {}", ids, e.getMessage());
            throw new RuntimeException("Erro interno ao processar
Work Items: " + e.getMessage(), e);
        }
    }

    @Override
    public List<IssueDto> searchIssues(String query) {
        if (query == null || query.trim().isEmpty()) {
            logger.debug("Query de busca vazia, retornando lista
vazia");
            return new ArrayList<>();
        }

        try {
            logger.debug("Executando busca de Work Items com query:
{}", query);

            // Constrói query WIQL (Work Item Query Language)
            String wiqlQuery = buildWiqlQuery(query);
```

```
            String url = String.format("%s?api-version=%s",
azureDevOpsProperties.getWorkItemQueriesApiUrl(),
azureDevOpsProperties.getApiVersion());

            HttpHeaders headers = createAuthHeaders();
            headers.setContentType(MediaType.APPLICATION_JSON);

            Map<String, String> requestBody = new HashMap<>();
            requestBody.put("query", wiqlQuery);

            HttpEntity<Map<String, String>> requestEntity = new
HttpEntity<>(requestBody, headers);

            ResponseEntity<String> response =
restTemplate.exchange(
                url, HttpMethod.POST, requestEntity, String.class);

            if (response.getStatusCode() == HttpStatus.OK &&
response.getBody() != null) {
                JsonNode queryResponse =
objectMapper.readTree(response.getBody());
                JsonNode workItemsNode =
queryResponse.get("workItems");

                if (workItemsNode != null &&
workItemsNode.isArray()) {
                    // Extrai IDs dos Work Items retornados pela
query
                    List<String> workItemIds = new ArrayList<>();
                    for (JsonNode workItemRef : workItemsNode) {
                        JsonNode idNode = workItemRef.get("id");
                        if (idNode != null) {
                            workItemIds.add(idNode.asText());
                        }
                    }

                    // Busca detalhes dos Work Items encontrados
                    if (!workItemIds.isEmpty()) {
                        return getIssuesByIds(workItemIds);
                    }
                }
            }

            logger.debug("Nenhum Work Item encontrado para a query:
{}", query);
            return new ArrayList<>();
```

```
        } catch (RestClientException e) {
            logger.error("Erro ao executar busca de Work Items com
query '{}': {}", query, e.getMessage());
            throw new RuntimeException("Erro na comunicação com
Azure DevOps: " + e.getMessage(), e);
        } catch (Exception e) {
            logger.error("Erro inesperado ao executar busca de Work
Items com query '{}': {}", query, e.getMessage());
            throw new RuntimeException("Erro interno ao processar
busca: " + e.getMessage(), e);
        }
    }

    /**
     * Mapeia um Work Item do Azure DevOps para o DTO comum
     */
    private IssueDto mapWorkItemToIssueDto(JsonNode workItem) {
        try {
            JsonNode fieldsNode = workItem.get("fields");
            if (fieldsNode == null) {
                logger.warn("Work Item sem campos (fields),
ignorando");
                return null;
            }

            IssueDto issue = new IssueDto();

            // ID
            JsonNode idNode = workItem.get("id");
            if (idNode != null) {
                issue.setId(idNode.asText());
            }

            // Título
            JsonNode titleNode = fieldsNode.get("System.Title");
            if (titleNode != null) {
                issue.setTitle(titleNode.asText());
            }

            // Estado
            JsonNode stateNode = fieldsNode.get("System.State");
            if (stateNode != null) {
                issue.setState(stateNode.asText());
            }

            // Tipo
            JsonNode typeNode =
fieldsNode.get("System.WorkItemType");
            if (typeNode != null) {
                issue.setType(typeNode.asText());
```

```
            }

            // Atribuído para
            JsonNode assignedToNode =
fieldsNode.get("System.AssignedTo");
            if (assignedToNode != null &&
assignedToNode.has("displayName")) {

issue.setAssignedTo(assignedToNode.get("displayName").asText());
            }

            // Data de criação
            JsonNode createdDateNode =
fieldsNode.get("System.CreatedDate");
            if (createdDateNode != null) {
                LocalDateTime createdDate =
parseAzureDateTime(createdDateNode.asText());
                issue.setCreatedDate(createdDate);
            }

            // URL
            JsonNode urlNode = workItem.get("url");
            if (urlNode != null) {
                // Constrói URL para visualização no Azure DevOps
                String viewUrl =
String.format("%s/%s/%s/_workitems/edit/%s",

azureDevOpsProperties.getBaseUrl(),

azureDevOpsProperties.getOrganization(),

azureDevOpsProperties.getProject(),
                                                issue.getId());
                issue.setUrl(viewUrl);
            }

            logger.trace("Work Item mapeado: {}", issue);
            return issue;

        } catch (Exception e) {
            logger.error("Erro ao mapear Work Item para IssueDto:
{}", e.getMessage());
            return null;
        }
    }

    /**
     * Cria headers de autenticação usando PAT com Basic Auth
     */
    private HttpHeaders createAuthHeaders() {
```

```
        HttpHeaders headers = new HttpHeaders();

        // Azure DevOps PAT usa formato: user:PAT em Base64
        String credentials = ":" +
azureDevOpsProperties.getPersonalAccessToken();
        String encodedCredentials = Base64.getEncoder()

.encodeToString(credentials.getBytes(StandardCharsets.UTF_8));

        headers.set(HttpHeaders.AUTHORIZATION, "Basic " +
encodedCredentials);
        headers.set(HttpHeaders.ACCEPT,
MediaType.APPLICATION_JSON_VALUE);

        return headers;
    }

    /**
     * Constrói uma query WIQL para busca de Work Items
     */
    private String buildWiqlQuery(String searchTerm) {
        // Query WIQL simples que busca no título e descrição
        return String.format(
            "SELECT [System.Id], [System.Title], [System.State],
[System.WorkItemType], " +
            "[System.AssignedTo], [System.CreatedDate] " +
            "FROM WorkItems " +
            "WHERE [System.TeamProject] = '%s' " +
            "AND ([System.Title] CONTAINS '%s' OR
[System.Description] CONTAINS '%s') " +
            "ORDER BY [System.CreatedDate] DESC",
            azureDevOpsProperties.getProject(),
            searchTerm,
            searchTerm
        );
    }

    /**
     * Converte string de data do Azure DevOps para LocalDateTime
     */
    private LocalDateTime parseAzureDateTime(String dateString) {
        try {
            // Azure DevOps retorna datas no formato ISO-8601 com
timezone
            // Ex: "2023-12-01T10:30:00.000Z"
            if (dateString.endsWith("Z")) {
                dateString = dateString.substring(0,
dateString.length() - 1);
            }
```

```
            if (dateString.contains(".")) {
                dateString = dateString.substring(0,
dateString.indexOf("."));
            }

            return LocalDateTime.parse(dateString,
DateTimeFormatter.ISO_LOCAL_DATE_TIME);
        } catch (Exception e) {
            logger.warn("Erro ao converter data '{}': {}",
dateString, e.getMessage());
            return LocalDateTime.now(); // Fallback para data atual
        }
    }
}
```

ANEXO D – AZUREDEVOPSPROPERTIES - INTEGRAÇÃO AZURE DEVOPS

Classe responsável pela integração com o Azure DevOps, conforme arquivo AzureDevOpsProperties.java.

```
package com.bugaware.config;

import
org.springframework.boot.context.properties.ConfigurationProperties
;
import org.springframework.context.annotation.Configuration;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

/**
 * Configurações do Azure DevOps para integração com Work Items
 *
 * As propriedades são carregadas do application.yml no namespace
'azure.devops'
 */
@Configuration
@ConfigurationProperties(prefix = "azure.devops")
public class AzureDevOpsProperties {

    @NotBlank(message = "Azure DevOps organization é obrigatória")
    private String organization;

    @NotBlank(message = "Azure DevOps project é obrigatório")
    private String project;

    @NotBlank(message = "Azure DevOps PAT é obrigatório")
    private String personalAccessToken;

    @NotBlank(message = "Azure DevOps base URL é obrigatória")
    private String baseUrl = "https://dev.azure.com";

    @NotNull(message = "Azure DevOps API version é obrigatória")
    private String apiVersion = "7.0";

    // Timeout para requisições em milissegundos
    private int timeoutMs = 30000;

    // Construtor padrão
    public AzureDevOpsProperties() {}

    // Getters e Setters
```

```
    public String getOrganization() {
        return organization;
    }

    public void setOrganization(String organization) {
        this.organization = organization;
    }

    public String getProject() {
        return project;
    }

    public void setProject(String project) {
        this.project = project;
    }

    public String getPersonalAccessToken() {
        return personalAccessToken;
    }

    public void setPersonalAccessToken(String personalAccessToken)
{
        this.personalAccessToken = personalAccessToken;
    }

    public String getBaseUrl() {
        return baseUrl;
    }

    public void setBaseUrl(String baseUrl) {
        this.baseUrl = baseUrl;
    }

    public String getApiVersion() {
        return apiVersion;
    }

    public void setApiVersion(String apiVersion) {
        this.apiVersion = apiVersion;
    }

    public int getTimeoutMs() {
        return timeoutMs;
    }

    public void setTimeoutMs(int timeoutMs) {
        this.timeoutMs = timeoutMs;
    }

    /**
```

```
     * Constrói a URL base para as APIs do Azure DevOps
     * @return URL base formatada
     */
    public String getApiBaseUrl() {
        return String.format("%s/%s/%s/_apis", baseUrl,
organization, project);
    }

    /**
     * Constrói a URL para Work Items API
     * @return URL para Work Items API
     */
    public String getWorkItemsApiUrl() {
        return String.format("%s/wit/workitems", getApiBaseUrl());
    }

    /**
     * Constrói a URL para Work Item Queries API
     * @return URL para Work Item Queries API
     */
    public String getWorkItemQueriesApiUrl() {
        return String.format("%s/wit/wiql", getApiBaseUrl());
    }

    @Override
    public String toString() {
        return "AzureDevOpsProperties{" +
                "organization='" + organization + '\'' +
                ", project='" + project + '\'' +                ",
personalAccessToken='[PROTECTED]'" +
                ", baseUrl='" + baseUrl + '\'' +
                ", apiVersion='" + apiVersion + '\'' +
                ", timeoutMs=" + timeoutMs +
                '}';
    }
}
```