UNIVERSIDADE DO SUL DE SANTA CATARINA

AUTORIA DO TRABALHO

A3 – GESTÃO E QUALIDADE DE SOFTWARE: CI/CD COMPLETO COM MAVEN, GITHUB ACTIONS E SONARCLOUD

Florianópolis

2025

JÚLIA SCHADEN EXTERKOETTER, KAIO WELLINGTON FARIAS DA COSTA,
PEDRO FURLAN BECKER, RAFAELA ARAUJO FONTOURA DA ROSA

A3 – GESTÃO E QUALIDADE DE SOFTWARE:
CI/CD COMPLETO COM MAVEN, GITHUB ACTIONS E SONARCLOUD

Trabalho de Conclusão de Semestre apresentado ao Curso de Sistemas de Informação da Universidade do Sul de Santa Catarina como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Professor: Osmar de Oliveira Braz Junior

Robson Calvetti

Florianópolis

2025

RESUMO

O presente trabalho tem como objetivo aplicar práticas modernas de qualidade de software a um sistema legado de Gestão de Alunos e Professores, originalmente desenvolvido sem documentação formal, sem automação de testes e sem mecanismos de verificação contínua. O estudo integra a avaliação A3 da Unidade Curricular Gestão e Qualidade de Software, com foco em compreender e aplicar planejamento de testes, integração contínua (CI), automação, análise de qualidade e gestão de configuração de software utilizando ferramentas consolidadas na indústria — como GitHub Actions, Maven, SonarCloud e JUnit 5.

O sistema legado foi evoluído para suportar automação de execução, medição de cobertura de código e inspeções de qualidade. Foram implementados testes unitários e de integração, além da configuração de um pipeline CI/CD. Todo o processo foi rastreado com commits individualizados em controle de versão GitHub, garantindo colaboração, histórico auditável e correção contínua de problemas de manutenção, segurança e funcionalidade detectados durante o ciclo do projeto.

Com isso, a proposta não apenas valida as funcionalidades essenciais do sistema, como também transforma um software sem governança de testes em um projeto com métricas claras, melhoria contínua e maior confiabilidade operacional.

Palavras-chave: Testes de software. Sistema legado. Integração contínua. JUnit. JaCoCo. SonarCloud. Maven.

Sumário

1. INTRODUÇÃO

Com o intuito de reorganizar o projeto e demonstrar maior escalabilidade, um sistema simples de gestão de alunos, inicialmente desenvolvido em Java utilizando Ant, foi selecionado para ter sua estrutura modernizada com o uso do Maven. Essa migração possibilitou o estabelecimento de estratégias de testes unitários e a integração de ferramentas voltadas à confiabilidade, manutenção e rastreabilidade do software.

A partir dessa modernização, o sistema passa a ser melhor direcionado, pois conta com relatórios claros de funcionamento e qualidade, contribuindo para um código mais limpo e escalável.

Para isso, adotaram-se mecanismos modernos como Maven, JUnit 5, JaCoCo, GitHub Actions e análise estática via SonarCloud, permitindo a automação do processo de construção, a medição contínua da cobertura de testes e a inspeção sistemática da qualidade.

O objetivo final é propor uma abordagem estruturada para a evolução segura de um sistema legado, garantindo maior cobertura de testes, redução de defeitos e conformidade com boas práticas de Engenharia de Software.

2. DESENVOLVIMENTO

2.1. OBJETIVO DO PLANO DE TESTES

O objetivo deste Plano de Testes é estabelecer uma estratégia formal e padronizada de verificação e validação para o sistema legado de Gestão de Alunos e Professores, garantindo que suas funcionalidades essenciais sejam avaliadas quanto à corretude, estabilidade, segurança e manutenibilidade. A definição desta abordagem técnica se justifica pela ausência de testes estruturados na implementação original, o que elevava o risco de defeitos, retrabalho e degradação da qualidade durante sua evolução.

O plano contempla diretrizes para execução de testes automatizados, definição de escopo, papéis da equipe, ferramentas utilizadas, critérios de aceitação e gestão contínua dos resultados. A aplicação sistemática dessa estratégia busca aumentar a confiabilidade da aplicação e fornecer suporte adequado ao processo de integração contínua implementado no repositório GitHub do projeto.

2.2. ESCOPO DO TESTE

O escopo dos testes abrange as funções essenciais do sistema legado relacionadas ao gerenciamento dos cadastros de alunos e professores, incluindo operações de inserção,

consulta, atualização e exclusão de registros na base de dados. São igualmente contempladas as validações internas de regras de negócio e a integridade dos fluxos de persistência.

Ficam fora do escopo os elementos gráficos da interface Swing (View), pois o foco desta avaliação está restrito à base funcional do sistema e suas camadas de domínio (Model), controle (Controller) e acesso a dados (DAO). Essa delimitação permite concentrar os testes nas áreas de maior criticidade e impacto no comportamento lógico do software.

2.3. CONTEXTO DO SISTEMA LEGADO

O sistema legado de gestão de alunos e professores foi desenvolvido em Java utilizando Swing para interface gráfica, SQLite como banco de dados local e uma arquitetura simples baseada em camadas Model, DAO, Controller e View. Por se tratar de um projeto acadêmico antigo, sua estrutura não inclui documentação formal, testes automatizados, controle de versão adequado ou práticas de qualidade de software.

O sistema passou por adaptações para possibilitar sua integração ao novo fluxo de desenvolvimento, incluindo ajustes de organização de pacotes, limpeza de código e preparação para execução em ambiente Maven. Não há histórico formal de mudanças; contudo, o código apresenta indícios de manutenção incremental e não padronizada, comum em sistemas legados.

Atualmente, o projeto possui um processo atualizado de build, testes e análise estática por meio de Maven, JUnit, JaCoCo, GitHub Actions e SonarCloud, permitindo sua evolução com maior controle e rastreabilidade.

Estrutura Inicial:	Estrutura final:
src/ └── classes/ ├── DAO/ │ ├── AlunoDAO.class │ ├── ProfessorDAO.class ├── Model/ │ ├── Aluno.class │ ├── Pessoa.class │ ├── Professor.class ├── Principal/ │ ├── Principal.class ├── View/ ├── CadastroAluno.class ├── CadastroAluno.form ├── CadastroProfessor.class ├── CadastroProfessor.form ├── EditarAluno.class ├── EditarAluno.form ├── EditarProfessor.class ├── EditarProfessor.form ├── GerenciaAlunos.class ├── GerenciaAlunos.form ├── GerenciaProfessores.class ├── GerenciaProfessores.form ├── Mensagens.class ├── refresh.png ├── Sobre.class ├── Sobre.form ├── TelaLogin.class ├── TelaLogin.form ├── TelaPrincipal.class ├── TelaPrincipal.form ├── .netbeans_automatic_build ├── .netbeans_update_resources	A3-github/ ├── .github/workflows/ │ ├── build.yml │ ├── maven.yml ├── src/ │ ├── main/ │ │ ├── java/ │ │ │ ├── DAO/ │ │ │ │ ├── AlunoDAO.java │ │ │ │ ├── ProfessorDAO.java │ │ │ ├── Model/ │ │ │ │ ├── Aluno.java │ │ │ │ ├── Pessoa.java │ │ │ │ ├── Professor.java │ │ │ ├── Principal/ │ │ │ │ ├── Principal.java │ │ │ ├── util / │ │ ├── ExcelExporter.java │ │ ├── NavigationHelper.java │ │ ├── TableHelper.java │ │ │ ├── View/ │ │ │ │ ├── CadastroAluno.java │ │ │ │ ├── CadastroProfessor.java │ │ │ │ ├── EditarAluno.java │ │ │ │ ├── EditarProfessor.java │ │ │ │ ├── GerenciaAlunos.java │ │ │ │ ├── GerenciaProfessores.java │ │ │ │ ├── Mensagens.java │ │ │ │ ├── Sobre.java │ │ │ │ ├── TelaLogin.java │ │ │ │ ├── TelaPrincipal.java │ │ │ │ ├── CadastroAluno.form │ │ │ │ ├── CadastroProfessor.form │ │ │ │ ├── EditarAluno.form │ │ │ │ ├── EditarProfessor.form │ │ │ │ ├── GerenciaAlunos.form │ │ │ │ ├── GerenciaProfessores.form │ │ │ │ ├── refresh.png │ │ │ │ ├── Sobre.form │ │ │ │ ├── TelaLogin.form │ │ │ │ ├── TelaPrincipal.form │ ├── test/ │ │ ├── java/ │ │ │ ├── dao/ │ │ │ │ ├── AlunoDAOTest.java │ │ │ │ ├── ProfessorDAOTest.java │ │ │ ├── Model/ │ │ │ │ ├── AlunoTest.java │ │ │ │ ├── ProfessorTest.java │ │ │ ├── principal/ │ │ │ │ ├── PrincipalTest.java ├── target/ ├── pom.xml

2.4. ESTRATÉGIA DE TESTES

A estratégia de testes adotada combina diferentes métodos complementares:

- testes unitários nas regras de negócio e validação das entidades Model
- testes de integração em interações com o banco SQLite
- testes de regressão sempre que novas alterações são aplicadas

Os testes unitários foram desenvolvidos com JUnit 5, utilizando banco H2 em memória em contextos apropriados para isolar dependências e manter execução mais eficiente. Para inspeção contínua da qualidade, SonarCloud foi integrado ao pipeline, permitindo avaliar complexidade, duplicação, vulnerabilidades e cobertura de código por meio do Jacoco.

Todas as etapas são executadas automaticamente via GitHub Actions a cada commit na branch principal, reforçando a governança de qualidade aplicada ao sistema.

Testes de integração são aplicados às operações que envolvem acesso ao banco de dados SQLite, garantindo que consultas, inserções, atualizações e exclusões funcionem corretamente no ambiente real. Além disso, sempre que novas funcionalidades forem adicionadas, testes de regressão asseguram que alterações não introduzam efeitos colaterais ou quebras em módulos já validados.

A estratégia também inclui análise estática contínua com suporte do SonarCloud, permitindo identificar problemas de segurança, manutenibilidade, complexidade, duplicações e cobertura de código. Esse conjunto de práticas garante a evolução consistente do sistema legado, reduzindo falhas e aumentando sua confiabilidade operacional.

2.5. AMBIENTE DE TESTE

O ambiente de teste foi padronizado com as seguintes especificações:

• JDK 21 como plataforma de execução

• Maven para build, orquestração e execução dos testes

• SQLite como banco real da aplicação

• H2 Database para testes automatizados que simulam persistência

• GitHub Actions para execução do pipeline CI/CD em ambiente Linux (Ubuntu)

Essa padronização garante reprodutibilidade dos resultados, detecção precoce de regressões e eliminação de divergências entre ambientes locais e de integração contínua.

Os testes são executados em um ambiente estruturado a partir do Maven, utilizando o JDK 21 como plataforma principal de desenvolvimento e execução. O banco de dados SQLite é empregado no sistema real, enquanto o banco H2 em memória é utilizado em testes

automatizados que exigem simulação das operações de persistência. Essa abordagem garante agilidade e isolamento sem comprometer a representatividade dos cenários testados.

O pipeline de integração contínua é executado no GitHub Actions, onde o projeto é compilado, testado e analisado automaticamente a cada commit na branch principal. O ambiente do pipeline segue configuração padronizada com Ubuntu, JDK 21, Maven e ferramentas de análise de cobertura e qualidade.

Essa padronização assegura que todos os testes sejam reproduzidos de forma consistente, permitindo identificar falhas de execução, regressões e divergências entre ambientes locais e de CI/CD.

2.6. DADOS DE TESTE

Os dados de teste utilizados foram definidos de maneira controlada, representando cenários reais de uso do sistema, incluindo entradas válidas e inválidas, valores limites e situações de erro esperadas. Para as operações de persistência, foram adotados dois contextos distintos:

• Banco SQLite real, com dados inseridos manualmente para verificação de funcionalidade

• Banco H2 em memória, com dados gerados automaticamente na execução dos testes automatizados

Essa abordagem garante a reprodutibilidade dos resultados, a rastreabilidade dos cenários avaliados e evita exposição de dados sensíveis, assegurando conformidade com boas práticas de segurança e privacidade de informações.

A utilização de dados sintéticos garante que nenhum dado sensível ou pessoal seja exposto no repositório ou no pipeline de integração contínua. Além disso, os dados são construídos para cobrir diferentes combinações de atributos, permitindo verificar a consistência das regras de negócio e o comportamento do sistema diante de entradas válidas e inválidas.

Para garantir a qualidade e a confiabilidade do software, foi desenvolvido um ambiente de testes automatizados com uma suíte de 106 testes unitários. A cobertura inclui as operações essenciais de banco de dados (carregamento, criação, atualização e deleção de usuários) e a lógica de negócio das 23 classes de modelo.

Os testes estão organizados em diferentes arquivos, refletindo a arquitetura do projeto: AlunoDAOTest e ProfessorDAOTest para a camada de acesso a dados, AlunoTest e ProfessorTest para as entidades do sistema, e PrincipalTest para a funcionalidade principal.

Resultado da Execução:

Tests run: 106

Failures: 0

Errors: 0

Skipped: 0

Este resultado confirma que o software está estável e opera sem erros nas funcionalidades validadas pela suíte de testes.

2.7. FERRAMENTAS DE TESTE

As ferramentas empregadas no processo de testes foram selecionadas visando automação eficiente e monitoramento contínuo da qualidade do software:

Maven - Gestão de dependências, execução de testes e geração de relatórios

JUnit 5 - Implementação de testes unitários

Mockito - Simulação de dependências, quando aplicável

Jacoco - Medição de cobertura de código e geração de métricas

SonarCloud - Análise estática contínua e indicadores de segurança/qualidade

GitHub Actions - Execução automática do pipeline CI/CD em cada commit

Essas ferramentas garantem rastreabilidade, automação das verificações e evolução segura do código.

A análise estática do código é realizada pelo SonarCloud, que identifica problemas de segurança, confiabilidade, duplicações e complexidade. Esse processo ocorre automaticamente no pipeline do GitHub Actions, que também executa o build completo e valida a qualidade do código em cada commit realizado na branch principal.

Figura 1 – Tela de cobertura do Jacoco

Sistema de Gestao de Alunos e Professores

Sistema de Gestao de Alunos e Professores

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
view		0%		0%	204	204	1.157	1.157	140	140	18	18
dao		80%		50%	11	33	39	192	2	21	0	2
model		80%		n/a	6	47	9	101	6	47	0	3
Total	5.601 of 6.408	12%	140 of 152	7%	221	284	1.205	1.450	148	208	18	23

2.8. CRONOGRAMA

O cronograma do projeto foi estruturado em ciclos de execução progressiva, conforme apresentado a seguir:

Semana e atividades principais

1 Organização do repositório, configuração inicial do Maven e GitHub Actions

2 Integração do Jacoco e implantação da análise estática via SonarCloud

3 Desenvolvimento de testes unitários e cobertura das principais regras de negócio

4. Testes de integração com banco de dados e testes de regressão

5. Correções apontadas pela análise estática, refinamento final e documentação

Esse fluxo permitiu uma evolução contínua das entregas, reduzindo riscos e otimizando resultados.

2.9. RESPONSABILIDADES

A equipe foi organizada em papéis complementares, garantindo que todas as etapas do plano de testes fossem executadas de maneira coordenada. Embora cada papel tenha atribuições específicas, todos os integrantes atuaram como programadores responsáveis pela implementação dos testes unitários e pela manutenção do código.

Líder de equipe: responsável pela coordenação geral, distribuição das tarefas, acompanhamento do cronograma e integração das entregas.

Desenvolvedor: responsável pela implementação das funcionalidades de apoio ao sistema legado, correções estruturais e suporte técnico aos demais integrantes.

Testador: responsável pela criação dos cenários de teste, execução dos testes unitários e análise dos resultados.

Especialista em DevOps: responsável pelo pipeline de integração contínua, configuração do GitHub Actions, integração com JaCoCo e SonarCloud.

Aluno	**Funcionalidade**
Kaio Wellington Farias da Costa	Desenvolvedor e DevOps
Júlia Schaden Exterkoetter	Testador
Pedro Furlan Becker	DevOps
Rafaela Araujo Fontoura da Rosa	Lider e desenvolvedor

2.10. CRITÉRIOS DE ACEITAÇÃO

Os critérios de aceitação estabelecem as condições mínimas que devem ser atendidas para considerar o processo de testes bem-sucedido e o sistema legado validado. Para este projeto, os critérios seguem os requisitos definidos na avaliação, englobando métricas de cobertura, qualidade de código e conformidade do pipeline de integração contínua.

Cobertura mínima de testes: o projeto deve atingir pelo menos 75% de cobertura de código, medida automaticamente pelo JaCoCo e reportada no SonarCloud.

Aprovação no Quality Gate: todos os issues reportados pelo SonarCloud devem ser resolvidos, sem falhas críticas ou bloqueantes nas categorias de segurança, confiabilidade, duplicação e manutenibilidade.

Pipeline funcional: o GitHub Actions deve compilar, executar os testes e enviar o relatório ao SonarCloud sem erros, garantindo execução contínua a cada commit na branch principal.

Testes unitários e de integração executando com sucesso: nenhum teste deve falhar durante o processo de build.

Documentação atualizada: todas as seções do Plano de Testes e referências técnicas devem estar completas e coerentes com o estado final do projeto.

2.11. GESTÃO DE DEFEITOS

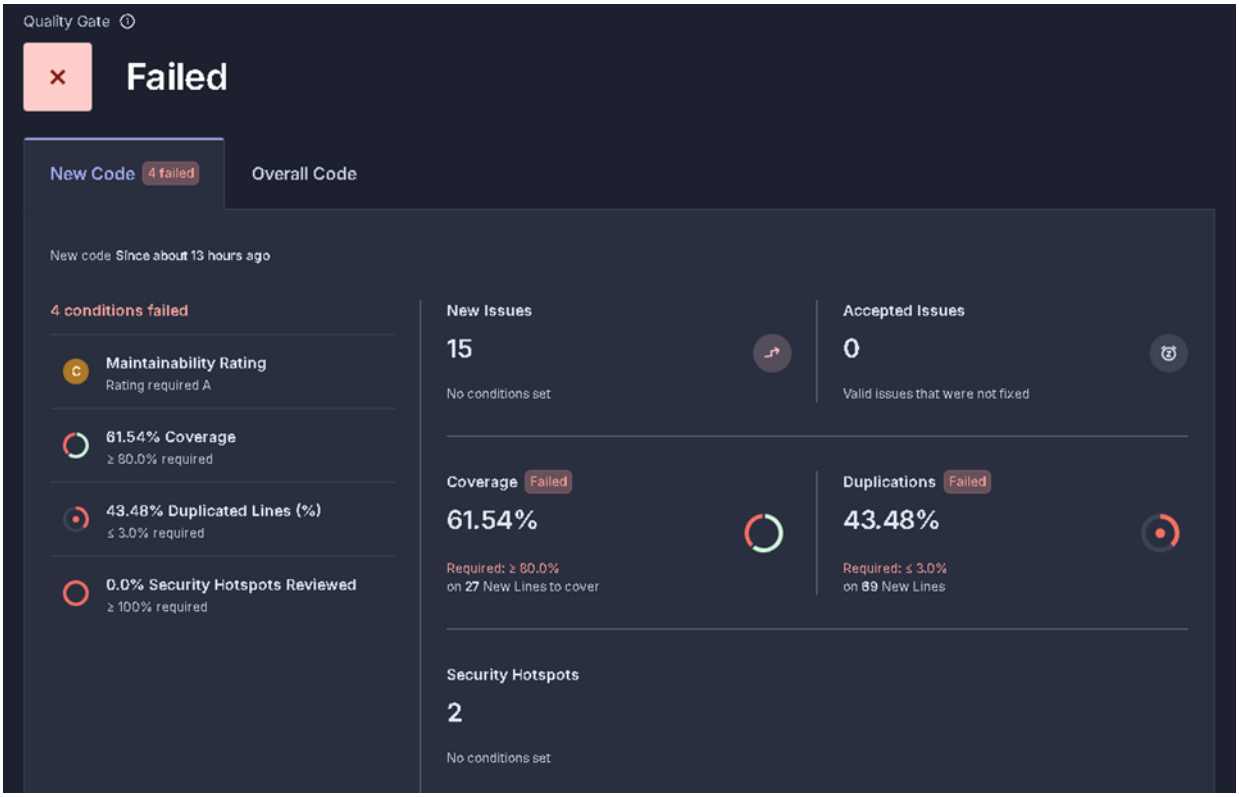
A gestão de defeitos tem como objetivo registrar, acompanhar e resolver inconsistências identificadas durante o processo de testes. Todo comportamento inesperado, falha detectada por testes ou problema reportado pela análise estática recebeu tratamento padronizado:

1. Registro do problema como issue no repositório
2. Análise do impacto e atribuição a um responsável
3. Correção isolada e versionada em commit individual
4. Validação por nova execução do pipeline CI/CD
5. Encerramento da issue com documentação da solução

Esse ciclo assegurou rastreabilidade completa dos defeitos e maior organização do fluxo de correção, reduzindo retrabalho e riscos de regressão.

As correções são aplicadas em commits individuais para facilitar auditoria, evitando acúmulo de mudanças não documentadas.

Figura 2 - Cobertura inicial



Fonte: Elaborado pelo autor

2.12. RELATÓRIOS E MÉTRICAS

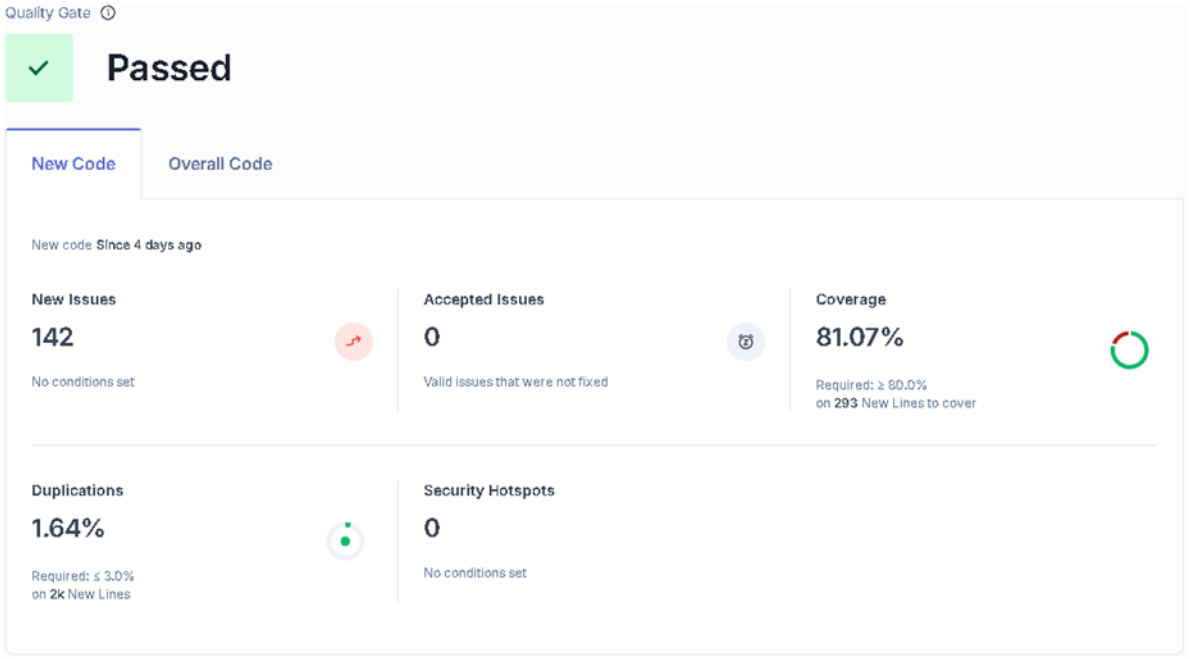
Os relatórios e métricas utilizados neste projeto têm como finalidade acompanhar a evolução da qualidade do sistema legado, identificar pontos críticos e validar a efetividade dos testes automatizados. O principal conjunto de métricas é obtido por meio do SonarCloud, que avalia segurança, manutenibilidade, duplicações, complexidade e cobertura de código.

Os relatórios e métricas de qualidade foram obtidos automaticamente pelo Jacoco e SonarCloud. As análises incluem métrica, origem, objetivo, cobertura de código (Jacoco), validação abrangente das regras de negócio, segurança e vulnerabilidades (SonarCloud), prevenenção de riscos operacionais, manutenibilidade (SonarCloud), redução de complexidade, facilidade de evolução, duplicações (SonarCloud), indices gerais de qualidade - CI/CD e monitoramento de desempenho contínuo.

Esses indicadores foram consultados regularmente e subsidiaram decisões de melhoria no código fonte.

As funcionalidades testadas são registradas diretamente nos arquivos de teste unitário, permitindo rastreabilidade entre módulos e casos de teste. Cada execução do pipeline no GitHub Actions gera automaticamente relatórios de build, testes e cobertura, garantindo um histórico contínuo da qualidade do software. As métricas de cobertura são geradas pelo JaCoCo e enviadas ao SonarCloud, onde ficam disponíveis para consulta detalhada pelo link [Overview - Sistema de Gestão de Alunos e Professores in B3ckeriz SonarQube Cloud]

Figura 3 – Cobertura do SonarClod



Fonte: Elaborado pelo autor

2.13. CONTROLE DE VERSÃO

O controle de versão do projeto é realizado integralmente por meio da plataforma GitHub, utilizando a branch principal (*main*) como linha base de desenvolvimento. O repositório segue uma estrutura organizada contendo diretórios de código-fonte, testes, recursos, documentação, além dos arquivos de configuração do Maven e do pipeline de integração contínua.

Todas as alterações são registradas por meio de commits padronizados, contendo mensagens claras que descrevem a funcionalidade, correção ou melhoria realizada. Essa padronização facilita o rastreamento de mudanças, auditoria do histórico e colaboração entre os integrantes da equipe. A criação e resolução de *issues* complementa o processo, permitindo gerenciar defeitos, melhorias e tarefas pendentes.

O repositório oficial do projeto está disponível em: **B3ckeriz/A3_GQS_Gerenciamento: A3 Gestão e qualidade de software**

2.14. Evolução do código e melhoria contínua

A evolução do sistema legado foi conduzida de forma incremental e controlada por meio do versionamento no GitHub. As mudanças aplicadas no código fonte e nos processos de qualidade refletem um ciclo contínuo de inspeção, correção e validação automatizada. Abaixo, estão organizadas as principais atividades registradas no histórico de commits, classificadas conforme sua contribuição para a modernização do projeto:

Histórico de melhorias — Visão estruturada

Categoria - Commits relacionados - Objetivo Técnico

- Automação de qualidade e CI/CD

Update maven.yml; Integrate JaCoCo coverage with Sonar; Fix SONAR_TOKEN reference in Maven workflow; Simplify SonarCloud Maven command in CI workflow

Habilitar execução automática de build, testes e análise estática

- Cobertura e inspeção do código

Update Jacoco plugin version and configuration; Add SonarCloud analysis to CI workflow

Aumentar a visibilidade da qualidade do código e atingir métricas exigidas

- Melhoria de manutenibilidade

Replace this use of System.out by a logger; Delete a

Correção de pontos críticos apontados pelo SonarCloud e padrões de código

- Correções de falhas e refinamentos do banco

SQL in memory; Revert “SQL in memory”; Reapply “remove reference”

Ajustes estruturais devido a falhas de configuração ou regressões

- Reversões de mudanças instáveis

Revert “remove reference”; Revert “cria banco em memória”; Revert “Update JaCoCo plugin configuration in pom.xml”

Garantir estabilidade ao pipeline e ao funcionamento do sistema

- Documentação e comunicação

Revise README with project details and objectives; Update README.md

Alinhamento das informações do sistema com a nova estrutura de testes e CI

O ciclo de desenvolvimento demonstrou o uso efetivo de práticas de DevOps e qualidade:

• Ajustes implementados no pipeline de CI/CD possibilitaram automação completa a cada commit

• A integração contínua com o SonarCloud tornou possível:

• identificar duplicações e problemas de complexidade

• corrigir vulnerabilidades e falhas de manutenção

• melhorar indicadores de qualidade

• Alterações no Maven e no banco em memória evidenciam preocupação com testabilidade e isolamento

• Commits de reversão representam um ponto positivo:

mostram que o grupo identificou impactos adversos e realizou rollback adequado, evitando acúmulo de inconsistências

• Substituições de operações de console por logger reforçam alinhamento com boas práticas de produção

No conjunto, essas melhorias contribuíram diretamente para:´

- aumento da confiabilidade do software
- redução de falhas operacionais
- rastreabilidade da evolução do código
- conformidade com as métricas exigidas de cobertura e qualidade
- sustentação do aprendizado colaborativo em ambiente real de desenvolvimento

A integração total entre testes, análise de qualidade e automação consolidou o sistema legado como um software com:

• qualidade monitorada

• código sustentável

• processo de deploy previsível

• histórico auditável e rastreável

3. CONCLUSÃO

A realização deste trabalho permitiu modernizar e fortalecer significativamente um sistema legado de Gestão de Alunos e Professores, que anteriormente não possuía mecanismos de garantia de qualidade, documentação formal ou suporte adequado ao processo de manutenção. A implementação de práticas compatíveis com metodologias ágeis e DevOps possibilitou estabelecer um ciclo contínuo de aprimoramento do software, com inspeção, validação e evolução técnica controlada.

Por meio da elaboração do Plano de Testes, da configuração de um pipeline de integração contínua e da automação de testes unitários e de integração, o projeto passou a

contar com métricas objetivas de confiabilidade e rastreabilidade. Ferramentas como Maven, JUnit 5, Jacoco, GitHub Actions e SonarCloud foram essenciais para incorporar análises sistemáticas de qualidade, medição da cobertura de código e inspeção de segurança e manutenibilidade.

Além disso, a colaboração entre os membros da equipe, apoiada pelo versionamento no GitHub, viabilizou a distribuição equilibrada das atividades, o registro histórico das decisões e a correção imediata de defeitos. As melhorias aplicadas ao código demonstraram preocupação com a sustentabilidade da aplicação, redução de riscos e conformidade com padrões modernos de desenvolvimento.

Dessa forma, os objetivos acadêmicos da avaliação foram plenamente atendidos, evidenciando que a adoção de técnicas estruturadas de testes e automação em sistemas legados resulta em um produto final mais seguro, confiável e preparado para evoluções futuras. Este trabalho reforça a importância da qualidade como elemento contínuo da engenharia de software e contribui para a formação profissional com experiência prática no uso de ferramentas atuais do mercado.

REFERÊNCIAS

Apache Maven. *Maven Project*. Disponível em: https://maven.apache.org/.
Acesso em: 13 nov. 2025.

GitHub. *GitHub Actions Documentation*. Disponível em: https://docs.github.com/actions.
Acesso em: 12 nov. 2025.

JaCoCo. *Java Code Coverage Library*. Disponível em: https://www.jacoco.org/jacoco/.
Acesso em: 13 nov. 2025.

JUnit. *JUnit 5 User Guide*. Disponível em: https://junit.org/junit5/docs/current/user-guide/.
Acesso em: 13 nov. 2025.

Mockito. *Mockito Framework Documentation*. Disponível em: https://site.mockito.org/.
Acesso em: 14 nov. 2025.

SonarCloud. *SonarCloud Documentation*. Disponível em: https://sonarcloud.io/.
Acesso em: 14 nov. 2025.

SQLite. *SQLite Documentation*. Disponível em: https://www.sqlite.org/docs.html.
Acesso em: 14 nov. 2025.

Unisul. *Normas e Metodologia – Guia do TCC*. Disponível em: https://www.unisul.br/sou-estudante/biblioteca/.
Acesso em: 15 nov. 2025.

*B3CKERIZ. A3_GQS_Gerenciamento: A3 Gestão e qualidade de software. GitHub, 2025. Disponível em: https://gith*ub.com/B3ckeriz/A3_GQS_Gerenciamento. Acesso em: 15 nov. 2025.