Implantação de Padrões e Práticas de Qualidade de Software: um estudo de caso

Daniel Paulo dos Santos[1], Rodrigo Miguel Tomazi[1], Jhony Maiki Maseto[1], Diego Fabio Schuh[1]

[1]Universidade Comunitária da Região de Chapecó - UNOCHAPECÓ

{daniel_paulo,rodrigo.tomazi,jhony,df_schuh}@unochapeco.edu.br

Abstract. *Poor software quality is a major industry challenge, negatively impacting productivity, maintenance and system sustainability. This article presents a case study on implementing software quality methodologies at a Higher Education Institution (HEI), with the goal of standardizing software engineering processes and improving product quality. The adopted methodology was action research, focusing on the creation and application of coding standards, versioning and code review practices, along with the increased use of automated testing and continuous documentation mechanisms. The adoption of practices such as Test-Driven Development (TDD) and Continuous Integration/Continuous Delivery (CI/CD) was considered a crucial factor in ensuring code quality and functionality. Previously scarce documentation became a strategic and dynamic resource, facilitating knowledge sharing. The results of a survey conducted with the development team show a significant increase in code readability, maintainability and traceability. There was a general perception of reduced defects and rework, as well as improved team communication. The HEI's experience underscores the importance of process standardization, team training and the adoption of appropriate tools to build a culture of continuous improvement, resulting in more robust and reliable software.*

Resumo. *A má qualidade de software é um dos maiores desafios da indústria, impactando negativamente a produtividade, manutenção e sustentabilidade dos sistemas. Este artigo apresenta um estudo de caso sobre a implementação de metodologias de qualidade de software em uma Instituição de Ensino Superior (IES), com o objetivo de padronizar processos de engenharia de software e elevar a qualidade dos produtos. A metodologia adotada foi a pesquisa-ação, com foco na criação e aplicação de padrões de codificação, versionamento e revisão de código, além da ampliação do uso de testes automatizados e da implementação de mecanismos de documentação contínua. A adoção de práticas como Test-Driven Development (TDD) e Continuous Integration/Continuous Delivery (CI/CD) foi vista como um fator crucial para garantir a qualidade e funcionalidade do código. A documentação, antes escassa, tornou-se um recurso estratégico e dinâmico, facilitando o compartilhamento de conhecimento. Os resultados da pesquisa, conduzida com a equipe de desenvolvimento, demonstram um aumento significativo na legibilidade, manutenibilidade e rastreabilidade do código. Houve uma percepção geral de redução de defeitos e retrabalhos, assim como uma melhoria na comunicação da equipe. A experiência da IES reforça a importância da padronização de processos, do treinamento da equipe e da adoção de ferramentas adequadas para construir uma cultura de melhoria contínua, resultando em software mais robusto e confiável.*

1. Introdução

A transformação digital tornou os computadores e seus sistemas essenciais para nossas vidas pessoais e profissionais. Para atender à demanda de armazenar, processar e

recuperar dados, esses sistemas precisam de *software* de qualidade. A qualidade de *software* está ligada diretamente ao valor do produto e é um atributo multidimensional e essencial, devendo funcionar de maneira confiável, ser seguro, fácil de usar e de manter, além de atender aos requisitos dos usuários e outras partes interessadas [Saini *et al.* 2020] [Kokol 2022].

A má qualidade em sistemas críticos pode ter consequências graves, como perdas financeiras, danos permanentes, falhas em missões ou até mesmo morte. Segundo a ISO (*International Organization for Standardization*) 8402:1986, a qualidade de *software* se refere a todas as características de um produto ou serviço que afetam sua capacidade de atender às necessidades dos usuários. A ISO 9001:2008 complementou essa definição, afirmando que a qualidade é determinada pela comparação das características de algo com um conjunto de requisitos [Ndukwe *et al.* 2022].

A qualidade de *software* está entre os maiores desafios da indústria de *software* atual. Apesar dos avanços das metodologias de desenvolvimento, as organizações ainda sofrem com dificuldades para garantir que seus sistemas atendam padrões de confiabilidade, desempenho e segurança. A ausência de processos padronizados, falta de documentação e negligência com testes automatizados contribuem para a introdução de defeitos, aumentando o custo de manutenção, uma vez que encontrar e corrigir um problema de *software* após a entrega pode ser 100 vezes mais caro do que encontrá-lo e corrigi-lo durante a fase de modelagem e desenvolvimento [Boehm e Basili 2005].

Neste contexto, o presente trabalho tem como objetivo principal descrever o processo de concepção, desenvolvimento e implementação de metodologias voltadas à padronização dos processos de engenharia de *software* em uma Instituição de Ensino Superior (IES). Os objetivos específicos incluem a definição e aplicação de padrões técnicos em todas as linguagens de programação utilizadas, a adoção de práticas de versionamento e revisão de código, a ampliação do uso de testes automatizados e a implementação de mecanismos de documentação contínua. Busca-se, com isso, elevar a qualidade dos produtos entregues, otimizar o tempo de manutenção e assegurar maior eficiência, rastreabilidade e sustentabilidade dos sistemas produzidos.

2. Trabalhos relacionados

Uma visão geral abrangente dos desafios e soluções em qualidade de *software* são oriundas de revisões sistemáticas da literatura e alguns estudos práticos. Estes estudos relacionados trazem boas práticas de codificação e documentação no âmbito de desenvolvimento de sistemas.

A dívida técnica, uma metáfora popularizada por Ward Cunningham, representa as escolhas de atalhos e decisões de *design* que levam a um *software* mais difícil de manter no futuro. Um mapeamento sistemático é feito para explorar a literatura sobre o tema, identificando as causas, tipos e estratégias para gerenciar a dívida técnica. Argumentos como a negligência com a qualidade de *software* é a principal causa desta dívida técnica, que, por sua vez, aumenta o custo e o tempo de manutenção, além de dificultar a implementação de novas funcionalidades [Li *et al.* 2015].

Métricas para qualidade de *software* é um dos aspectos importantes no desenvolvimento de novos *softwares.* Uma revisão sistemática da literatura para

identificar e categorizar as métricas mais utilizadas em ambientes de desenvolvimento ágil são destacadas neste estudo. Embora o foco da metodologia ágil seja entregas rápidas, a qualidade do *software* continua sendo um desafio e requer métricas adequadas para ser monitorada. Métricas essas como satisfação do cliente, confiabilidade do código e manutenibilidade mostram como o gerenciamento contínuo da qualidade é importante para o sucesso de equipes ágeis [Muneer e Saquib 2020].

Estudos como automação de testes avaliam o impacto da implementação de testes automatizados na qualidade de seus produtos. A automação de testes reduz significativamente o número de defeitos introduzidos no *software*, diminui o tempo de atrasos e aumenta a confiança nas entregas. Evidências empíricas mostram que o investimento em testes automatizados são estrategicamente eficazes para garantir a qualidade, reforçando a sua introdução em ambientes de desenvolvimento [Eldhose e Joy 2021].

Além de tudo isso, a documentação de um *software* é indispensável para alterações futuras e, muitas vezes, o próprio código pode servir como tal. Nesse sentido, a nomenclatura de identificadores, como variáveis, constantes, funções, *tags*, classes e objetos, são importantes, especialmente para projetos em equipe, tornando o código-fonte altamente legível e fácil de manter [Wang *et al.* 2010].

Os trabalhos relacionados evidenciam que métricas, padrões, estratégias e automatização de testes são fatores importantes para a qualidade de *software*. Sendo assim, este estudo de caso aplicou um conjunto de procedimentos, padrões e ferramentas para melhoria na transcrição e documentação de códigos, não se limitando a uma única metodologia.

3. Metodologia

Esta pesquisa é verificada por meio de um estudo de caso, de natureza aplicada. Dessa forma, a questão de pesquisa é abordada por meio da metodologia de pesquisa-ação. A Figura 1 ilustra as 6 etapas utilizadas para aplicação da metodologia deste estudo.

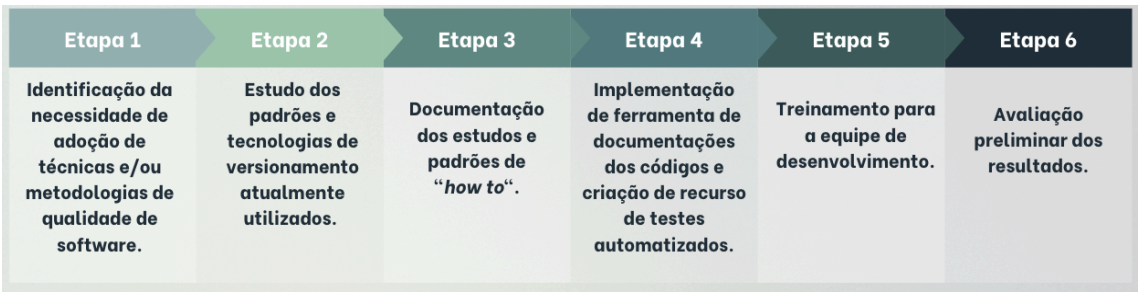

Figura 1. Etapas da metodologia.

Inicialmente, na etapa 1, a organização identificou a necessidade de elevar a qualidade de suas aplicações, impulsionada por fatores como a busca por redução de *bugs*, a melhora da manutenibilidade e o aumento da satisfação do cliente. A partir disso, a etapa 2 trata da análise dos processos e ferramentas existentes, focando no estudo dos padrões de codificação e nas tecnologias de versionamento já em uso pela

equipe, permitindo entender o cenário atual e identificar as áreas que podem ser aprimoradas pela nova metodologia.

Na sequência, a etapa 3 já foca em documentar os estudos e os novos padrões de como fazer, passo fundamental para formalizar as diretrizes e garantir que todos na equipe sigam as mesmas práticas. A etapa 4 é a fase de implementação, onde são introduzidas ferramentas de documentação técnica para os códigos e criados recursos para rodar testes automatizados, garantindo que a qualidade do código seja mantida de forma consistente e automatizada, reduzindo a carga de trabalho manual da equipe.

A etapa 5 concentra-se no treinamento da equipe, essencial para garantir a adesão e o entendimento da nova metodologia. E por fim, na etapa 6, é realizada uma avaliação preliminar dos resultados. Essa análise de dados permite medir o impacto das mudanças e tomar decisões informadas para a implementação em larga escala e a longo prazo.

4. Identificação da necessidade de adoção de metodologias de qualidade de *software*

Durante a análise inicial do fluxo de desenvolvimento da IES, identificou-se um cenário marcado pela ausência de padronização na esteira produtiva de codificação. Cada desenvolvedor adotava seu próprio estilo de programação, definindo convenções de nomenclatura, estrutura de código e formatação de maneira arbitrária. Essa heterogeneidade impactava diretamente a legibilidade e manutenibilidade dos sistemas, dificultando a comunicação e colaboração entre diferentes membros da equipe nos projetos.

Observou-se ainda que o processo de revisão de código apresentava fragilidade significativa. Em diversos casos, o *code review* simplesmente não era realizado e as alterações eram submetidas sem qualquer descrição ou contextualização, dificultando a compreensão das propostas de alterações no código-fonte. Quando a revisão acontecia, mostrava-se deficiente pela falta de critérios uniformes, onde cada revisor utilizava parâmetros subjetivos para avaliar o código, resultando em revisões inconsistentes.

Muitas vezes a aprovação do código ocorria mais pela urgência ou pressão de atender rapidamente às demandas do que pela verificação efetiva de qualidade e conformidade com boas práticas de desenvolvimento. O que não somente comprometia a detecção precoce de defeitos, mas também eliminava o potencial do processo de revisão como ferramenta de aprendizado e aprimoramento técnico para a equipe.

Esse contexto evidenciou a necessidade urgente de estabelecer diretrizes claras e compartilhadas, capazes de alinhar a produção de código aos padrões comuns e reduzir as variações que comprometem a qualidade final do *software*.

5. Estudos e documentação de padrões e tecnologias

A definição dos padrões técnicos e metodológicos foi o resultado de uma combinação da experiência empírica da equipe de desenvolvimento em seu dia a dia, com um estudo de referências consolidadas que orientam a escrita técnica e a padronização de processos. Entre essas, destaca-se a RFC (*Request for Comments*) 2119 [Bradner 1997],

que estabelece diretrizes para o uso de termos normativos em documentos técnicos, garantindo clareza, precisão e consistência na comunicação.

5.1. Padrões de Codificação

Padrões de codificação consistem em diretrizes que asseguram consistência estrutural, facilitando leitura, compreensão e colaboração entre programadores [Wang *et al.* 2010]. Em equipes com alta rotatividade ou que terceirizam parte do desenvolvimento, tais padrões assumem papel essencial, na medida em que possibilitam a rápida assimilação e manutenção da base de código por novos integrantes da equipe.

No contexto da IES, a definição desse padrão foi apoiada em boas práticas consolidadas no mercado, como a PSR-12 [Szanto 2025], amplamente reconhecida pela comunidade *PHP-FIG* como referência de estilo e formatação de código em projetos PHP, assim como recomendações de Windler e Daubois (2022).

A partir do diagnóstico realizado e das fragilidades identificadas no processo de desenvolvimento, o primeiro passo adotado foi a criação de um conjunto formal de padrões de codificações específicos para as linguagens utilizadas pelo time da IES, armazenado em um repositório centralizado, ilustrado na Figura 2, facilitando o acesso pela equipe.

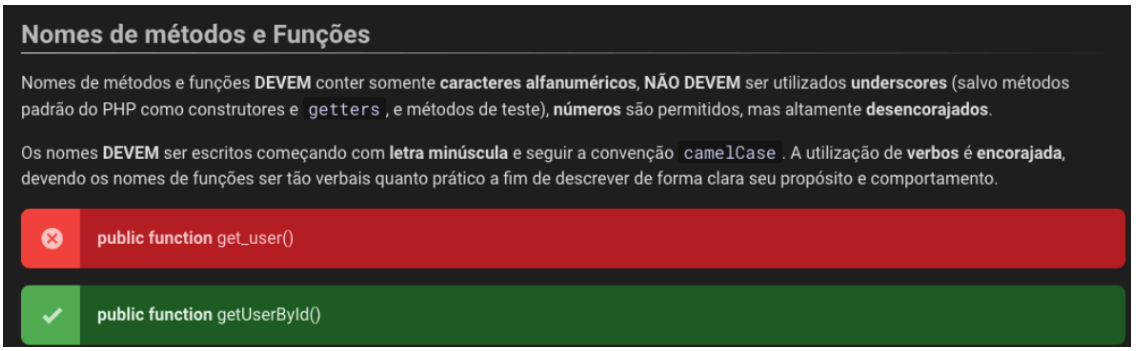

Figura 2. Exemplo da documentação de padrões de codificação

A padronização passou a abranger desde as convenções de nomenclatura, até práticas recomendadas de organização e escrita de código, contando com exemplos e trechos de código para exemplificação, garantindo consistência entre os projetos e facilitando a colaboração entre membros da equipe. Além disso, foram estabelecidos procedimentos claros para atualização e aprimoramento contínuo dessa documentação, incluindo a submissão de propostas por qualquer integrante do time e a revisão obrigatória por desenvolvedores seniores, assegurando que as alterações estivessem alinhadas às boas práticas e à realidade técnica da organização.

5.2. Padrões de versionamento

O controle de versionamento do código-fonte, viabilizado por ferramentas como o Git, constitui um pilar fundamental no ambiente de desenvolvimento da IES, pois garante a rastreabilidade das alterações, organiza o histórico evolutivo dos projetos e possibilita a contribuição estruturada e colaborativa entre os membros da equipe.

Para que o fluxo de codificação e versionamento acontecesse da maneira mais eficiente possível, foi estabelecido um conjunto de diretrizes documentadas que

definem, de maneira clara, o fluxo de contribuição via MR (*Merge Requests)*, aliado a um processo formal de *code review*.

Essas diretrizes contemplam desde à criação de *branches* com nomenclatura padronizada, vinculada ao identificador da tarefa que gerou a alteração no código, até o uso de mensagens de *commit* seguindo a especificação *Conventional Commits*, garantindo legibilidade e rastreabilidade do histórico de alterações. Além disso, os MRs devem conter descrições completas e contextualizadas, registrando não apenas o que foi alterado, mas também o motivo e as decisões técnicas adotadas. Essa prática visa transformar o MR em um artefato de documentação viva, útil para consultas futuras apoiando a manutenção do sistema.

O processo de *code review* também foi padronizado e documentado, sendo instituído como etapa obrigatória antes da integração das mudanças à aplicação em produção. As normas desse processo adotam o princípio de que não existe código perfeito, apenas código melhor, de modo que um MR promova melhorias relevantes, devendo ser aprovado por um analista, mesmo que não atinja perfeição absoluta.

Essa metodologia assegura a conformidade com os padrões definidos, além de detectar inconsistências e fomentar o aprendizado contínuo entre desenvolvedores. Dessa forma, o processo de versionamento passou a atuar não somente como controle de mudanças, mas também como mecanismo efetivo de melhoria contínua da qualidade do *software*.

5.3. Testes

Embora a IES já possuísse testes unitários implementados, constatou-se que a metodologia atual para criação destes se mostrava ineficiente na cobertura das regras de negócio, assim como não possuía padronização ou ferramentas para execução automática dos mesmos.

O primeiro passo adotado foi a padronização da escrita dos testes, que serviu como um dos balizadores para a melhoria contínua da qualidade do *software* na IES. Foi adotado o padrão de nomenclatura *GIVEN–WHEN–THEN* [Fowler 2013], o que tornou o propósito e o escopo de cada teste mais claro e autoexplicativo, facilitando a leitura e compreensão por qualquer membro da equipe. Toda a padronização dos testes foi documentada da mesma forma dos padrões de código e utilizando o mesmo repositório.

Na estruturação dos testes a principal técnica aplicada foi o TDD (*Test-Driven Development)*, uma abordagem de desenvolvimento na qual os testes são escritos antes da implementação do código. Assim a equipe passou a compreender que essa abordagem não se resume à verificação de funcionalidade, mas atua como um método de projeto. Ao escrever os testes antes da implementação, os desenvolvedores são levados a tomar decisões de análise e arquitetura que favorecem a coesão e o baixo acoplamento do código, resultando em uma solução mais fácil de testar e consequentemente mais ágeis de manter e evoluir.

Ainda como parte dos esforços para fortalecer a governança da qualidade de *software*, foi implementado uma *pipeline* de *Continuous Integration/Continuous Delivery* (CI/CD) integrado ao repositório de código da IES, atuando em duas frentes principais:

1. **Existência dos testes unitários:** sempre que um *commit* é submetido a um MR, o sistema identifica se os arquivos adicionados ou modificados possuem testes correspondentes. Caso algum elemento não esteja coberto, é aberta automaticamente uma discussão no MR notificando o autor sobre a ausência, permitindo a correção antes da aprovação, conforme exemplificado na Figura 3.

Figura 3. Exemplo de discussão aberta em um MR

2. **Execução automática dos testes unitários:** sempre que um *commit* é submetido a um MR, todos os testes relacionados aos arquivos modificados ou atualizados são executados, gerando relatórios detalhados, conforme Figura 4, indicando os casos de testes que passaram e os que falharam, acompanhados das respectivas mensagens de erro e rastreamento.

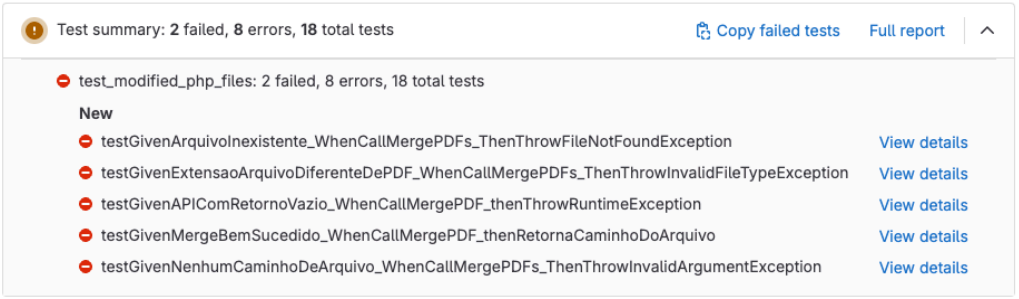

Figura 4. Relatório de testes unitários de um MR

Com a aplicação dessas melhorias relacionadas aos testes unitários, observou-se que os mesmos se tornaram recursos estratégicos para "documentação viva" do projeto. Ao contrário de registros por escrito, que tendem a ser esquecidos e estarem desatualizados com o tempo, os testes, caso continuamente executados e atualizados, acabam servindo como especificações técnicas das regras de negócio, assim como descrevem, de forma inequívoca, o comportamento esperado do *software*.

5.4. Documentação

Antes da aplicação de padronização de processos descrita neste artigo, a documentação dos sistemas na IES apresentava problemas significativos, pois muitas vezes não era produzida e quando era, não existia um repositório centralizado para disponibilizá-la à equipe, dificultando assim a consulta e utilização desses registros. Como consequência, o conhecimento técnico ou de negócio ficavam restritos aos profissionais envolvidos diretamente no projeto.

Com a implantação dos novos padrões, foi utilizada a mesma estrutura que foram descritos os padrões de codificação, exemplificados na Figura 2, para a criação de documentações internas, visando não contemplar apenas o código, mas também aspectos da funcionalidade e utilização do sistema, além de sua arquitetura e fluxos de processos, complementando a legibilidade do código ao explicitar seu propósito e comportamento esperado. Dessa forma e devido à facilidade em escrever ou encontrar a documentação, ela se tornou uma referência estratégica no compartilhamento de conhecimento entre os membros da equipe.

Para assegurar a qualidade e atualização contínua da documentação, passou-se a exigir sua criação ou atualização para que um MR seja aprovado, garantindo que a documentação reflita o estado atual do sistema e facilitando tanto a manutenção quanto à evolução futura.

6. Treinamento da equipe

A implantação das novas práticas e padrões na IES exigiu uma etapa de treinamento, visando alinhar a equipe de desenvolvimento às metodologias propostas e garantir sua aplicação consistente no cotidiano dos projetos. O processo foi organizado em duas etapas principais, considerando a diversidade de experiência técnica dos integrantes.

A primeira etapa contemplou sessões teóricas, realizadas em reuniões presenciais, nas quais foram apresentados os conceitos de qualidade de *software*, as motivações para a padronização e o funcionamento dos novos fluxos de trabalho. Também foram ostentados os novos padrões de codificação, versionamento, *code review* e documentação, destacando exemplos práticos que evidenciam problemas anteriores e como essas novas práticas os mitigariam.

Já a segunda etapa, envolveu um processo de acompanhamento contínuo nas semanas subsequentes à implantação. Desenvolvedores com mais experiência atuaram como professores, revisando o trabalho dos demais com foco educativo, promovendo *feedbacks* e sanando dúvidas ocasionais. Esse treinamento, embora simples, possibilitou uma adoção mais fluida das mudanças pelos desenvolvedores, assim como levantou assuntos e criou espaços para troca de conhecimento entre os integrantes da equipe, fortalecendo a cultura e colaboração no setor de desenvolvimento da IES.

7. Resultados

Para avaliar os efeitos dos novos métodos de qualidade de *software*, aplicou-se uma pesquisa *online* com a equipe de desenvolvimento da IES. O questionário continha 15 perguntas obrigatórias, cujas respostas seguiram uma escala Likert de 5 pontos, variando de “concordo totalmente” (5) a “discordo totalmente” (1). Para cada uma das perguntas, também foi adicionada um campo opcional para que o profissional pudesse explicar o motivo da sua resposta, caso ele considerasse relevante. A Figura 5 ilustra a avaliação geral da equipe na percepção de melhora no desenvolvimento de sistema após a implementação dos novos padrões de qualidade.

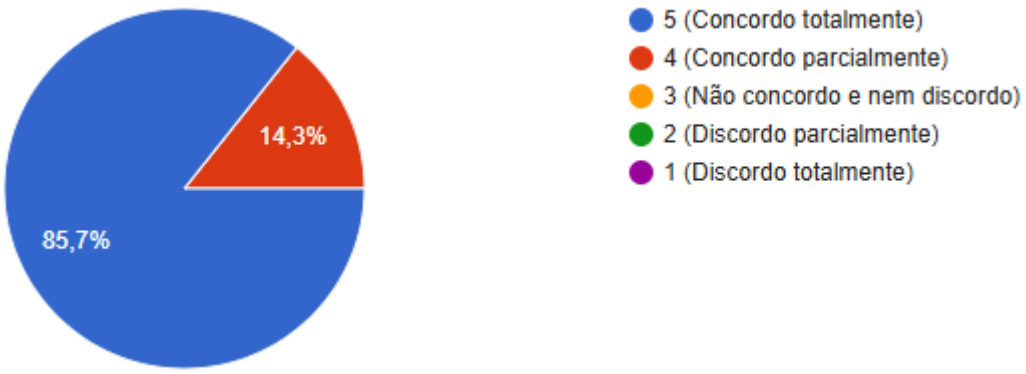

Figura 5. Avaliação geral da metodologia aplicada

Conforme os resultados iniciais da pesquisa, houve um aumento significativo na qualidade dos *softwares* desenvolvidos após a aplicação dos novos padrões de qualidade, já que todas as implementações seguiam um mesmo padrão, tornando-se mais legíveis. Além disso, 71,5% da equipe percebeu uma redução no número de problemas ou retrabalhos em relação ao antigo cenário, destacando que os padrões de codificação facilitam a manutenção e a compreensão do código e que, quando combinadas a boas práticas e com uma cultura sólida de testes, são essenciais para a redução de problemas, onde 85,7% da equipe percebeu que a execução automática de testes via CI/CD trouxe mais segurança nas implantações. Ainda, 71,4% da equipe percebeu uma melhora na comunicação interna, pois o uso de uma padronização à torna mais clara e objetiva, facilitando discussões técnicas e de regras de negócio.

8. Conclusão

A necessidade crescente por *software* de alta qualidade na era da transformação digital impõe desafios significativos para as organizações. A falta de padronização, a negligência para com os testes e documentação, como observado na IES em questão, resulta em sistemas de difícil manutenção, que elevam os custos operacionais e comprometem a satisfação do usuário.

O presente estudo de caso teve como objetivo central analisar a aplicação de metodologias e padrões orientados à qualidade de software. Os resultados evidenciam que a adoção sistemática de metodologias, em conjunto com a implementação de testes automatizados e de um processo contínuo de documentação, contribui de forma significativa para a transformação de contextos de desenvolvimento desorganizados em ambientes colaborativos e produtivos. Destacam-se, nesse processo, a padronização de práticas de codificação, o estabelecimento de um modelo estruturado de versionamento e a incorporação de testes unitários fundamentados em TDD, integrados a pipelines de CI/CD. Essas medidas revelaram impactos positivos concretos, promovendo maior legibilidade, manutenibilidade e rastreabilidade do código, além de consolidar uma base mais robusta para a evolução do sistema.

Os resultados da pesquisa interna confirmam o sucesso da iniciativa. A equipe reportou maior facilidade na manutenção dos sistemas, uma redução nos defeitos e retrabalhos, além de uma melhora na comunicação e no aprendizado mútuo. A

documentação, que antes era escassa e descentralizada, passou a ser vista como um recurso estratégico e dinâmico, essencial para o compartilhamento de conhecimento.

A qualidade de *software* não é um atributo opcional, mas um pilar essencial para o desenvolvimento de produtos digitais robustos e confiáveis. A padronização de processos, aliada ao treinamento da equipe e ao uso de ferramentas adequadas, cria uma cultura de melhoria contínua que beneficia tanto a organização quanto os usuários.

9. Referências

Boehm, Barry; Basili, Victor R. (2005) Software defect reduction top 10 list. Foundations of empirical software engineering: the legacy of Victor R. Basili, v. 426, n. 37, p. 426-431, 2005.

Bradner, S. O. (1997) Key words for use in RFCs to Indicate Requirement Levels. RFC 2119. RFC Editor, 1997. DOI: 10.17487/RFC2119. Disponível em: https://www.rfc-editor.org/info/rfc2119. Acesso em: 18 ago. 2025.

Eldhose, J., Joy, T. (2021). The impact of automated testing on software quality: A case study. International Journal of Modern Trends in Engineering and Research, 1-6.

Fowler, M.. (2013). Given When Then. Disponível em: https://martinfowler.com/bliki/GivenWhenThen.html. Acessado em: 18 ago. 2025.

Janzen, D.; Saiedian, H. (2005) Test-driven development concepts, taxonomy, and future direction. Computer, v. 38, n. 9, p. 43-50, 2005.

Kokol, P. (2022). Software Quality: How Much Does It Matter? Electronics, v. 11, n. 16, p. 2485, 10 ago. 2022

Li, Z., Ma, M., Yang, B. (2015). Technical debt: A systematic mapping study. Journal of Software Engineering and Applications, 653-662.

Muneer, Z., Saqib, S. (2020). A systematic literature review on software quality metrics in agile development. Journal of Computer Science, 1735-1748.

Ndukwe I. G., Licorish S. A., Tahir A., MacDonell S. G. (2022). How have views on Software Quality differed over time? Research and practice viewpoints. Journal of Systems and Software, p. 111524, out. 2022.

Saini G. L., Panwar D., Kumar S., Singh V., (2020). A systematic literature review and comparative study of different software quality models. Journal of Discrete Mathematical Sciences and Cryptography, v. 23, n. 2, p. 585–593, 17 fev. 2020.

Szanto, K. (2019) Extended Coding Style Guide. Disponível em: https://www.php-fig.org/psr/psr-12/. Acessado em: 18 ago. 2025.

Wang Y., Wang S., Li X., Li H., Du J.(2010). Identifier naming conventions and software coding standards: A case study in one school of software. In: 2010 International Conference on Computational Intelligence and Software Engineering. IEEE, 2010. p. 1-4.

Windler, C., & Daubois, A. (2022). Clean Code in PHP: Expert tips and best practices to write beautiful, human-friendly, and maintainable PHP. Packt Publishing Ltd.