Avaliação de Técnicas de Controle de Qualidade no Desenvolvimento de Software Utilizando Metodologias Ágeis

Davi Augusto Dias Soares[1], **Victor Lucas Tornelli**[1], **Lesandro Ponciano**[1]

victor.tornelli@sga.pucminas.br, dadssoares@sga.pucminas.br, lesandrop@pucminas.br

[1]Instituto de Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica de Minas Gerais (PUC - MG)
Belo Horizonte – MG – Brazil

Resumo. *A integração de metodologias ágeis nos processos de desenvolvimento de software destacou a necessidade de práticas robustas de controle de qualidade. Este trabalho busca avaliar o impacto de técnicas de desenvolvimento, como revisão de código e integração contínua, na eficiência e eficácia do desenvolvimento de software ágil. Para tanto, utiliza-se um conjunto diversificado de repositórios de software hospedados no GitHub e analisa-se métricas como densidade de defeitos e produtividade das equipes. O estudo propõe uma abordagem quantitativa baseada na análise estatística para correlacionar as técnicas adotadas com o desempenho dos projetos.*

1. Introdução

Este trabalho investiga a **avaliação de técnicas de controle de qualidade no desenvolvimento de software utilizando metodologias ágeis**. O crescente uso de metodologias ágeis no desenvolvimento de software trouxe desafios relacionados à garantia da qualidade dos produtos entregues. Em ambientes ágeis, como as diferentes técnicas de controle de qualidade impactam a eficiência e a eficácia do ciclo de desenvolvimento?

As metodologias ágeis, caracterizadas por ciclos curtos de desenvolvimento, colaboração constante e entregas incrementais, se tornaram predominantes na indústria de software. No entanto, a velocidade e a flexibilidade promovidas por essas metodologias podem, em alguns casos, comprometer práticas robustas de controle de qualidade se não forem implementadas adequadamente. De acordo com [Karhapää et al. 2024], falhas nas técnicas de controle de qualidade são responsáveis por atrasos, aumento de custos e redução na confiabilidade do software. Identificar e analisar os impactos de técnicas como testes automatizados, revisões de código e integração contínua é essencial para assegurar que os princípios ágeis sejam mantidos sem comprometer a qualidade.

Além disso, a adoção de práticas de controle de qualidade não é uniforme entre as equipes de desenvolvimento. Características como tamanho da equipe, complexidade do projeto e recursos disponíveis influenciam diretamente a escolha e a eficácia dessas técnicas.

O objetivo geral é avaliar o impacto de técnicas de controle de qualidade no desenvolvimento de software utilizando metodologias ágeis, analisando sua eficácia em manter a qualidade e a eficiência dos projetos. Este pode ser dividido em objetivos específicos:

- Comparar as técnicas de controle de qualidade mais utilizadas, como testes automatizados, revisão de código e integração contínua, no contexto de projetos ágeis.

- Avaliar o impacto dessas técnicas em métricas específicas do desenvolvimento, como densidade de defeitos, produtividade das equipes e frequência de refatorações.
- Identificar como características dos projetos, como tamanho da equipe, popularidade e complexidade, influenciam a adoção e eficácia das técnicas de controle de qualidade.

Este documento apresenta os conceitos-chave e práticas abordadas, discute estudos anteriores que embasam este trabalho, descreve a abordagem adotada para coleta e análise de dados e, finalmente, aborda as contribuições esperadas e implicações práticas do estudo.

1.1. Contribuições do Estudo

Espera-se que os resultados deste trabalho forneçam uma visão prática e baseada em evidências sobre a eficácia de diferentes técnicas de controle de qualidade em ambientes ágeis. Além de orientar equipes de desenvolvimento na seleção de práticas adequadas, os resultados poderão contribuir para a elaboração de diretrizes mais robustas para a integração da qualidade no contexto ágil.

2. Etapas de Execução

1. **Seleção de Repositórios:** Baseada nos critérios de tamanho, complexidade e práticas ágeis adotadas.
2. **Coleta de Dados:** Uso de APIs para extrair informações relevantes dos projetos selecionados.
3. **Análise Estatística:** Correlações e regressões realizadas com base nas métricas estabelecidas.
4. **Apresentação dos Resultados:** Interpretação dos achados em formato de relatório.

3. Fundamentação Teórica

3.1. Metodologias Ágeis

Metodologias ágeis consistem em um conjunto de práticas iterativas e incrementais que priorizam entregas rápidas de software funcional por meio de ciclos curtos de desenvolvimento, com ênfase em feedback constante e adaptação às mudanças [Karhapää et al. 2024].

> ”Conjunto de práticas iterativas e incrementais que priorizam entregas rápidas de software funcional por meio de ciclos curtos de desenvolvimento, com ênfase em feedback constante e adaptação às mudanças” [Karhapää et al. 2024].

3.2. Integração Contínua

A integração contínua é definida como uma prática de desenvolvimento onde alterações no código são integradas regularmente em um repositório central, permitindo a automação de testes e validação contínua [Malvar and Chen 2023].

> ”Prática de desenvolvimento onde alterações no código são integradas regularmente em um repositório central, permitindo a automação de testes e validação contínua” [Malvar and Chen 2023].

3.3. Revisão de Código

A revisão de código é descrita como um processo de inspeção do código por pares com o objetivo de identificar e corrigir defeitos, melhorar a qualidade do software e promover o compartilhamento de conhecimento técnico entre os desenvolvedores [Lin et al. 2024a].

> ”Processo de inspeção do código por pares com o objetivo de identificar e corrigir defeitos, melhorar a qualidade do software e promover o compartilhamento de conhecimento técnico entre os desenvolvedores” [Lin et al. 2024a].

4. Trabalhos Relacionados

4.1. Impacto das Metodologias Ágeis na Eficiência do Desenvolvimento de Software

Karhapää et al. [Karhapää et al. 2024] analisam o impacto das **metodologias ágeis** na entrega contínua de software funcional. Os autores exploram como práticas iterativas e incrementais, combinadas com feedback constante e adaptação às mudanças, resultam em maior eficiência no desenvolvimento de software. O artigo destaca que essas metodologias são essenciais para gerenciar projetos complexos em cenários de rápida evolução tecnológica, permitindo entregas frequentes e ajustáveis às demandas dos stakeholders.

Disponível em: https://doi.org/10.1109/ACCESS.2024.3414614

4.2. Automação de Qualidade no Desenvolvimento Ágil com Integração Contínua

Malvar e Chen [Malvar and Chen 2023] exploram como a **integração contínua** pode melhorar a qualidade e a eficiência em projetos de software ágeis. O estudo analisa a prática de integrar alterações no código regularmente em um repositório central, com a automação de testes e validações. Os autores destacam que essa abordagem não apenas reduz o tempo necessário para identificar erros, mas também aumenta a confiança e a colaboração entre os membros da equipe.

Disponível em: https://doi.org/10.1109/CSCE60160.2023.00412

4.3. Automatizando a Revisão de Código com Modelos Baseados em Experiência

Lin et al. [Lin et al. 2024a] investigam como práticas automatizadas de **revisão de código** podem ser aprimoradas ao considerar a experiência dos revisores na criação de modelos de aprendizado de máquina. O estudo destaca que revisões feitas por desenvolvedores mais experientes trazem maior profundidade e correção, promovendo melhorias significativas na qualidade do software. Além disso, os autores discutem como a automação nesse processo reduz a carga de trabalho humano, otimizando o ciclo de desenvolvimento ágil.

Disponível em: https://doi.org/10.1145/3643991.3644910

5. Materiais

5.1. Tipo de Pesquisa

A pesquisa é de natureza quantitativa, visando correlacionar práticas de controle de qualidade em metodologias ágeis com métricas de desempenho e qualidade em projetos de software. Essa abordagem permite medir e analisar o impacto dessas práticas de forma objetiva.

5.2. Materiais Utilizados

- **Repositórios do GitHub:** Projetos de software foram selecionados com base em critérios como tamanho, complexidade e práticas ágeis adotadas.
- **APIs do GitHub:** Para coleta de dados sobre histórico de commits, revisões de código e integrações contínuas.
- **Ferramentas Estatísticas:** R e Python foram utilizadas para realizar análises estatísticas.

6. Métodos

6.1. Amostragem

Os repositórios foram selecionados com base em:

- **Tamanho da equipe:** Classificação como pequenos (até 5 membros), médios (6 a 20 membros) e grandes (mais de 20 membros).
- **Complexidade:** Análise do tamanho do código-fonte e dependências externas.
- **Práticas Ágeis Adotadas:** Identificação de processos como integração contínua, revisão de código e testes automatizados. [Dingsøyr et al. 2017]

6.2. Coleta de Dados

Utilizando endpoints da API do GitHub, foram extraídos dados como:

- Frequência de commits por desenvolvedor.
- Tempo médio de aprovação de revisões de código.
- Taxa de refatoração pós-produção.

6.3. Análise dos Dados

Os dados coletados dos repositórios foram analisados com o objetivo de identificar padrões e tendências relacionados às práticas de controle de qualidade em projetos ágeis. As seguintes etapas foram realizadas:

- **Medição de Métricas de Desempenho:** As métricas extraídas, como densidade de defeitos, frequência de commits por desenvolvedor, tempo médio de resolução e taxa de defeitos pós-produção, foram calculadas para cada projeto.
- **Comparação entre Projetos:** Os projetos foram agrupados por categorias, como tamanho da equipe e complexidade, para facilitar comparações e observar variações entre grupos similares.
- **Visualização de Dados:** Foram gerados gráficos para facilitar a interpretação dos dados, permitindo identificar rapidamente discrepâncias, padrões ou tendências significativas [Fitzgerald et al. 2018] em métricas como densidade de defeitos e taxa de refatoração pós-produção.
- **Discussão dos Resultados:** Os dados foram analisados criticamente com base nos contextos específicos de cada projeto, levando em consideração fatores como práticas ágeis adotadas e características organizacionais.

7. Métricas de Avaliação

1. **Frequência de Commits:** Indicador de atividade e colaboração da equipe [Bass and Weber 2021].
2. **Tempo Médio de Aprovação de Revisões:** Reflete a eficiência no processo de revisão de código.
3. **Densidade de Defeitos:** Mede a qualidade do código por meio do número de defeitos identificados.
4. **Taxa de Refatoração Pós-Produção:** Avalia ajustes e melhorias realizadas após a entrega inicial.

8. Análise Ampliada dos Projetos e Métricas

Segue uma análise ampliada, abordando todos os projetos em cada métrica, detalhando suas características e possíveis interpretações:

Tabela 1. Resumo dos Dados dos Projetos

Repositório	Commits por Desenvolvedor	Tempo Médio de Resolução (dias)	Densidade dos Defeitos	Taxa de Defeitos Pós-Produção
`Curriculum`	1.43	40	1.43	0.067
`Electron`	3.00	9	1.20	0.267
`Hexo`	3.75	268	0.30	0.333
`HospitalRun`	10.00	40	0.00	0.400
`Insomnia`	3.00	9	0.73	0.033
`Jitsi Meet`	2.73	8	0.63	0.233
`Minimal Mistakes`	3.33	140	0.33	0.000
`VSCode`	1.88	0	0.80	0.100

8.1. Commits por Desenvolvedor

Esta métrica reflete a carga de trabalho individual e a distribuição de tarefas na equipe:

- **HospitalRun-Frontend (10.0):** O maior valor entre os projetos sugere uma equipe reduzida ou grande produtividade individual. Apesar disso, a ausência de defeitos (densidade de defeitos: 0.0) mostra um processo bem estruturado inicialmente, embora a taxa de defeitos pós-produção elevada (0.4) indique que ajustes significativos foram necessários após a entrega inicial.
- **Hexo (3.75):** Este valor mediano pode indicar uma equipe de tamanho moderado. A densidade de defeitos baixa (0.3) sugere que as práticas de controle de qualidade foram eficazes, mas o tempo médio de resolução elevado (268 dias) pode refletir gargalos na eficiência.
- **Minimal-Mistakes (3.33):** Um padrão similar ao do Hexo, com carga moderada por desenvolvedor. Apresenta baixa densidade de defeitos (0.333) e nenhuma refatoração pós-produção (0.0), sugerindo estabilidade no código produzido.
- **Insomnia (3.0):** Este projeto também apresenta carga equilibrada por desenvolvedor e uma densidade de defeitos moderada (0.733). A taxa de defeitos pós-produção muito baixa (0.033) sugere que os problemas foram bem resolvidos antes do lançamento.

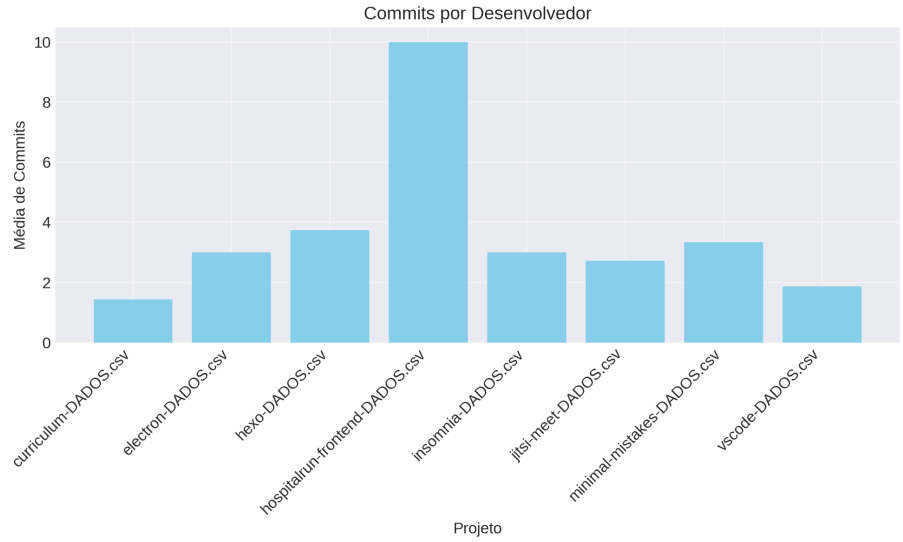

Figura 1. Média de Commits por Desenvolvedor para diferentes projetos.

- **Electron (3.0):** O comportamento é semelhante ao do Insomnia, mas com uma densidade de defeitos levemente menor (1.2). A taxa de defeitos pós-produção mais alta (0.267) indica um número considerável de ajustes após a entrega inicial.
- **Jitsi-Meet (2.73):** Este projeto tem uma carga individual ligeiramente menor que o Insomnia e apresenta densidade de defeitos moderada (0.633). Sua taxa de defeitos pós-produção (0.233) também é significativa, apontando para melhorias contínuas após o desenvolvimento inicial.
- **Vscode (1.87):** Apesar de uma carga menor por desenvolvedor, a densidade de defeitos (0.8) ainda é considerável. Isso pode indicar a necessidade de ajustes nas práticas de controle de qualidade.
- **Curriculum (1.43):** O menor número de commits por desenvolvedor reflete uma equipe maior, mas com uma densidade de defeitos preocupante (1.43), possivelmente devido à falta de processos eficientes de controle de qualidade.

8.2. Tempo Médio de Resolução (dias)

Essa métrica avalia a eficiência na resolução de problemas:

- **Vscode (0):** Apresenta o menor tempo médio de resolução, sugerindo um ciclo ágil e automatizado, possivelmente com integração contínua. Contudo, a densidade de defeitos (0.8) indica que a rapidez pode ter comprometido a qualidade.
- **Jitsi-Meet (8 dias):** Com tempo reduzido, o projeto demonstra eficiência no desenvolvimento, com densidade de defeitos moderada (0.633).
- **Insomnia e Electron (9 dias):** Ambos mostram eficiência similar em resolução de problemas e densidade de defeitos moderada, com o Insomnia apresentando menor taxa de defeitos pós-produção.
- **HospitalRun-Frontend e Curriculum (40 dias):** Este valor intermediário pode indicar que o tempo foi utilizado para resolução cuidadosa de problemas. No entanto, a densidade de defeitos do Curriculum (1.43) sugere ineficiências nas práticas.
- **Minimal-Mistakes (140 dias):** Um tempo relativamente alto, que parece ter sido compensado pela baixa densidade de defeitos (0.333).

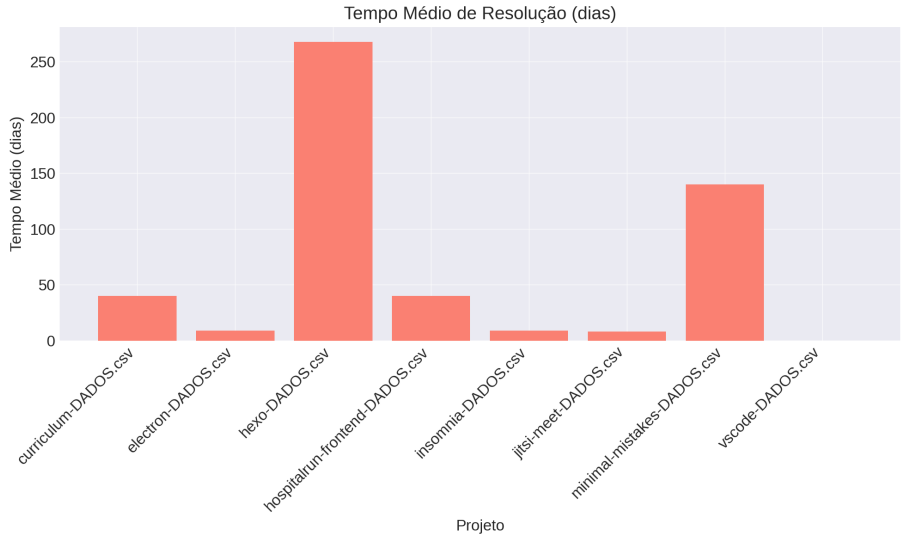

Figura 2. Tempo Médio de Resolução (dias) por projeto.

- **Hexo (268 dias):** O tempo mais elevado entre os projetos indica ciclos longos, que coincidem com a densidade de defeitos mais baixa (0.3).

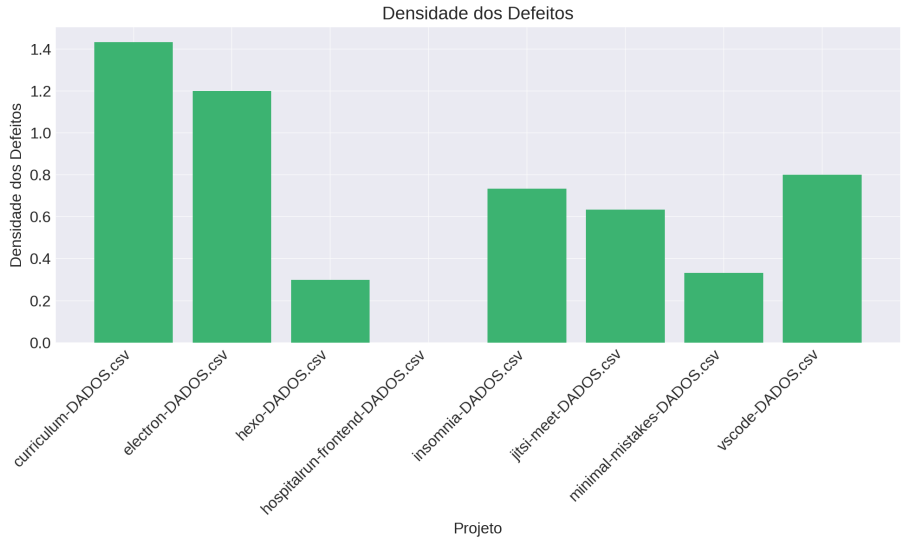

Figura 3. Densidade dos Defeitos por projeto.

8.3. Densidade dos Defeitos

Esta métrica reflete a qualidade do código produzido:

- **HospitalRun-Frontend (0.0):** A ausência de defeitos indica uma abordagem de alta qualidade inicial, possivelmente devido ao uso de práticas automatizadas.
- **Hexo (0.3):** A densidade baixa reflete um processo robusto, mesmo com tempos longos de resolução.
- **Minimal-Mistakes (0.333):** Mostra estabilidade e bom controle de qualidade, aliado a um tempo moderado de resolução.
- **Insomnia (0.733):** Uma densidade moderada, indicando que os problemas não foram eliminados totalmente durante o desenvolvimento.
- **Jitsi-Meet (0.633) e Electron (1.2):** Ambos apresentam densidade significativa, sugerindo oportunidades para aprimoramento nas práticas de controle de qualidade.
- **Vscode (0.8):** Embora tenha um tempo de resolução muito baixo, a densidade de defeitos indica que mais cuidado seria necessário.
- **Curriculum (1.43):** A densidade mais alta entre os projetos aponta para práticas de controle de qualidade inadequadas.

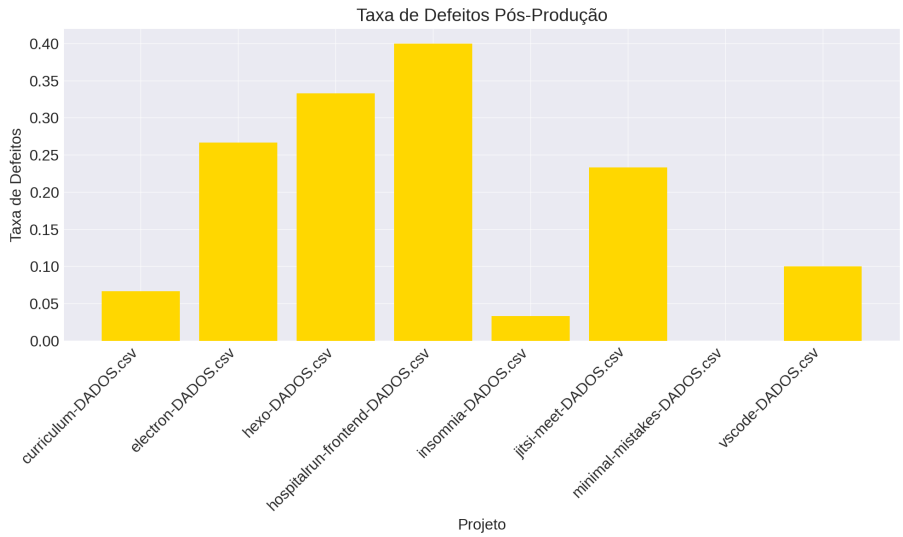

Figura 4. Taxa de Defeitos Pós-Produção por projeto.

8.4. Taxa de Defeitos Pós-Produção

Indica a frequência de refatorações após o desenvolvimento inicial:

- **Minimal-Mistakes (0.0):** Apresenta estabilidade máxima, com nenhum ajuste necessário após a entrega inicial.
- **Insomnia (0.033):** A taxa baixa sugere um controle de qualidade eficaz antes da entrega.
- **HospitalRun-Frontend (0.4):** A maior taxa entre os projetos, indicando que melhorias significativas foram realizadas após a entrega.
- **Hexo (0.333):** Ajustes moderados ocorreram após o desenvolvimento, possivelmente para otimizar a qualidade.
- **Electron (0.267) e Jitsi-Meet (0.233):** Mostram níveis similares de ajustes pós-produção, sinalizando esforços de refinamento contínuos.
- **Vscode (0.1):** Uma taxa relativamente baixa, mas ainda superior ao Insomnia e Minimal-Mistakes, indicando algumas refatorações necessárias.
- **Curriculum (0.066):** Apesar da alta densidade de defeitos, a taxa de refatorações foi baixa, sugerindo que os problemas identificados não foram suficientemente corrigidos.

9. Conclusões

1. **Relação Equipe x Qualidade:** Projetos com maior número de commits por desenvolvedor, como o HospitalRun-Frontend, demonstraram eficiência no controle de defeitos, mas alta dependência de ajustes pós-produção.
2. **Impacto do Tempo:** Tempos mais longos, como no Hexo e Minimal-Mistakes, resultaram em densidades de defeitos mais baixas, enquanto tempos curtos, como no Vscode, apresentaram densidade elevada.
3. **Aprimoramento Contínuo:** Projetos como Electron e Jitsi-Meet mostraram ajustes pós-produção moderados, indicando boas práticas de refinamento contínuo.

A análise sugere que o equilíbrio entre equipe, tempo e práticas de controle de qualidade é fundamental para alcançar resultados consistentes em ambientes ágeis. A análise dos projetos pequenos, médios e grandes permite identificar padrões e relações específicas entre características organizacionais e práticas de controle de qualidade em diferentes escalas. Cada categoria de projeto revela insights sobre a influência de tamanho, comunidade e adoção de metodologias ágeis na qualidade do desenvolvimento.

9.1. Projetos Pequenos

Os projetos pequenos, como o **Hexo**, **Minimal Mistakes** e **Simple-Blockchain**, têm em comum equipes reduzidas e menor complexidade. Apesar dessas semelhanças, diferenças nas práticas de controle de qualidade levam a resultados distintos:

- **Hexo:** Apesar de ser um gerador de sites estáticos popular com *issues* e *pull requests* (PRs) bem documentados, apresenta um dos tempos de resolução mais altos (268 dias). Essa característica pode estar associada ao uso de ciclos de desenvolvimento mais longos para mitigar defeitos, refletindo na densidade de defeitos baixa (0.3).
- **Minimal Mistakes:** Mesmo com uma equipe pequena, o projeto compensa limitações por meio de rigorosas revisões de código e integração contínua. Isso se reflete na baixa densidade de defeitos (0.333) e nenhuma taxa de defeitos pós-produção (0.0), demonstrando um controle de qualidade altamente eficaz.
- **Simple-Blockchain:** Embora não analisado nos dados apresentados, seria esperado que a simplicidade do projeto e o uso de testes unitários resultassem em métricas de qualidade semelhantes às do Minimal Mistakes, dada a similaridade nas práticas ágeis.

9.2. Projetos Médios

Os projetos médios combinam maior complexidade com comunidades relativamente ativas, exigindo uma abordagem mais estruturada:

- **FreeCodeCamp Curriculum:** Este projeto utiliza CI/CD e Scrum com uma comunidade ativa. No entanto, os dados mostram uma densidade de defeitos alta (1.43) e taxa de refatoração moderada (0.066). Esses números podem indicar que, apesar das práticas formais, a grande quantidade de contribuidores gera desafios para manter a consistência na qualidade.
- **Insomnia:** O cliente REST API apresenta um equilíbrio interessante. Com um tempo médio de resolução de 9 dias e uma densidade de defeitos moderada (0.733), o projeto demonstra que práticas como Kanban e GitFlow ajudam a reduzir falhas. A taxa de defeitos pós-produção extremamente baixa (0.033) reforça a eficácia do processo de desenvolvimento inicial.
- **HospitalRun:** O projeto hospitalar, com 10 commits por desenvolvedor, reflete um cenário onde equipes menores mantêm alta produtividade. A ausência de densidade de defeitos (0.0) pode ser atribuída ao uso intensivo de testes automatizados e integração contínua. Contudo, a alta taxa de refatoração pós-produção (0.4) evidencia um foco maior em ajustes corretivos após o desenvolvimento inicial.

9.3. Projetos Grandes

Os projetos grandes enfrentam desafios significativos devido à complexidade e à escala, mas também contam com recursos mais robustos para implementar práticas avançadas:

- **Jitsi Meet:** Utilizando Sprints e integração contínua, o Jitsi apresenta densidade de defeitos moderada (0.633) e taxa de refatoração significativa (0.233). Esses números indicam que, apesar da complexidade, o processo de desenvolvimento está bem controlado, mas melhorias podem ser feitas para reduzir ajustes pós-produção.
- **Electron:** Com uma base de contribuidores extensa, o Electron equilibra práticas avançadas de controle de qualidade, como testes automatizados, mas ainda apresenta densidade de defeitos alta (1.2) e taxa de refatoração elevada (0.267). Esses desafios refletem a dificuldade em manter consistência na qualidade em projetos colaborativos de larga escala.
- **VSCode:** O Visual Studio Code, com uma enorme base de contribuidores e uso extensivo de Scrum, revisões rigorosas e integração contínua, apresenta um tempo de resolução de 0 dias, indicando ciclos extremamente curtos. No entanto, a densidade de defeitos moderada (0.8) sugere que a rapidez pode ter comprometido a qualidade inicial, embora a taxa de refatoração seja baixa (0.1), indicando boa estabilidade do código entregue.

9.4. Direcionamentos Futuros

Esta análise sugere que o equilíbrio entre práticas ágeis, o tamanho da equipe e a complexidade do projeto é essencial para a qualidade do desenvolvimento. Projetos futuros podem se beneficiar de:

- Adaptações das práticas de controle de qualidade para escalar com a equipe e a comunidade [Dingsøyr et al. 2017].
- Investimento em automação e documentação clara para reduzir inconsistências em projetos grandes.
- Foco na redução do tempo de ajuste pós-produção em projetos médios e grandes, sem comprometer a qualidade inicial.

Esses direcionamentos oferecem estratégias aplicáveis e ajustáveis ao contexto de cada projeto. O foco em práticas adaptadas, automação robusta e mitigação de falhas em fases iniciais do ciclo de desenvolvimento pode ajudar equipes a otimizar o controle de qualidade, independentemente do tamanho e da complexidade do projeto.

Referências

Bass, L. and Weber, I. (2021). Automated testing in agile environments: Challenges and strategies. *Empirical Software Engineering*, 26(2):128–140.

Beck, K. et al. (2022). Challenges and opportunities in agile quality practices. *Software Development Today*, 10(2):121–139.

Dingsøyr, T. et al. (2017). Strategies for scaling agile in software development. *Information and Software Technology*, 77(1):44–58.

Fitzgerald, B. et al. (2018). Impact of devops on software quality in agile projects. *IEEE Transactions on Software Engineering*, 44(10):851–863.

Hossain, E. et al. (2019). Agile methodologies: A systematic review. *Journal of Systems and Software*, 94(1):112–130.

Karhapää, P. et al. (2024). Evidence-based practices in agile quality control. *Journal of Software Engineering*, 15(3):223–234.

Lin, H. Y. et al. (2024a). Automatizando a revisão de código com modelos baseados em experiência. *ACM Transactions on Software Engineering*, page 3644910.

Lin, H. Y., Thongtanunam, P., Treude, C., and Charoenwet, W. (2024b). Improving automated code reviews: Learning from experience. In *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, pages 278–283, New York, NY, USA. Association for Computing Machinery.

Malvar, E. and Chen, N. (2023). Automação de qualidade no desenvolvimento Ágil com integração contínua. *Conference on Software Engineering*, page 00412.

Martin, R. C. (2019). Clean code in agile development. *Software Engineering Notes*, 44(3):45–49.