QUALIDADE DE DESENVOLVIMENTO DE SOFTWARE EM PARTE DA REGIÃO NOROESTE DO ESTADO DE SÃO PAULO COM FOCO NA MICRORREGIÃO DE JALES

QUALITY OF SOFTWARE DEVELOPMENT IN PART OF THE NORTHWEST REGION OF THE STATE OF SÃO PAULO FOCUSING ON THE JALES MICROREGION

Eduardo S. Silva[1], Hiágor H. Vieira[2], Jorge L. Gregório[3]

[1]Faculdade de Tecnologia Prof. José Camargo – Fatec Jales, eduardo.silva218@fatec.sp.gov.br
[2]Faculdade de Tecnologia Prof. José Camargo – Fatec Jales, hiagor.vieira@fatec.sp.gov.br
[3]Faculdade de Tecnologia Prof. José Camargo – Fatec Jales, jorge.gregorio@fatec.sp.gov.br

Informação e Comunicação
Subárea: Engenharia e Desenvolvimento de Software

RESUMO

A maioria das empresas de desenvolvimento de sistemas informatizados possuem falhas nos processos do ciclo de vida de desenvolvimento de software (SLDC - Software Development Life Cycle). Nesse contexto, o presente trabalho possui como objetivo compreender como as empresas de desenvolvimento de software do noroeste paulista, principalmente na microrregião de Jales, gerenciam os processos relacionados ao SLDC. Para isso, foi conduzida uma pesquisa com empresas da região citada, além de um levantamento bibliográfico sobre as melhores práticas que envolvem os processos do SLDC. A pesquisa evidenciou que a maioria das empresas não seguem as boas práticas, nem adotam padrões para melhorar o processo de desenvolvimento de software, impactando negativamente em diversos aspectos dentro da organização.
Palavras-chave: engenharia de software; desenvolvimento de software; qualidade de código; sistemas legados; metodologias ágeis.

ABSTRACT

Most computer systems development companies have problems in the Software Development Life Cycle (SLDC) processes. In this context, the present work aims to understand how software development companies in the northwest of São Paulo, especially in the micro-region of Jales, has been manage processes related to SLDC. For this, a survey was conducted with companies in the that region, in addition to a bibliographic survey on the best practices of the SLDC processes. The research showed that most companies really do not follow good practices, nor adopt standards to improve the software development process, causing negative impacts in several aspects within the organization.
Keywords: software engineering; software development; code quality; legacy systems; agile methodologies.

1 INTRODUÇÃO

Segundo Pressman e Maxim (2021), o *software* é a tecnologia que revolucionou a sociedade atual. De fato, o mundo moderno é, em muitos aspectos, gerenciado por ferramentas tecnológicas que, em essência, executam algoritmos definidos em linguagens computacionais, além de prover uma comunicação quase que em tempo real na maioria das aplicações. Assim, o *software* tem sido objeto de discussão desde os primórdios da era moderna da computação, principalmente ao considerar seu desenvolvimento ou construção.

Nesse contexto, o bom desenvolvedor é o profissional capaz de transformar especificações formais em *softwares* que atendam às necessidades do usuário e do negócio. Entretanto, desenvolver *softwares* não é uma tarefa trivial, principalmente considerando a qualidade do código, pois exige do desenvolvedor raciocínio lógico, visão crítica acerca de tecnologias e plataformas, conhecimento de domínio, testes, entender melhores práticas e ter competências socioemocionais. Ademais, é necessário aprender com os erros do passado para se adaptar no presente e moldar um novo futuro.

Porém, as coisas não são bem assim. Em 1970, com a crise de *software*, já existiam problemas de códigos sem qualidade e processos inadequados, o que gerava grandes problemas entre cliente e empresa e causava um alto prejuízo para as partes envolvidas, pois, quando o código está ruim, é difícil implementar uma nova funcionalidade, o tempo de resolução dos problemas sobe consideravelmente, o que prejudica mentalmente os desenvolvedores e impacta negativamente o negócio que depende daquele sistema.

Assim, surgiu a Engenharia de Software (ES), que buscava se preocupar com técnicas, ferramentas e processos, ou seja, em criar uma abordagem sistemática sobre o desenvolvimento com o objetivo de tentar melhorar as coisas. De fato, a ES ajudou, entretanto, muitos problemas daquela época ainda continuaram (PRESSMAN; MAXIM, 2021).

De acordo com Stripe (2018), os prejuízos de com código ruim representa um custo aproximado de 85 bilhões de dólares anualmente, além de uma média de 41.1 horas da semana dos desenvolvedores dedicados a resolver débitos técnicos e tratar problemas de má qualidade. Um outro relatório, de autoria de Krasner (2021), mostra que o custo com a má qualidade de código foi, apenas nos EUA, de 2.08 trilhões de dólares. Deste valor, cerca de 1.56 trilhões em falhas em operações, contando as de segurança, e 260 bilhões em projetos sem sucesso, além de 520 bilhões em sistemas legados.

Krasner (2021) não inclui o custo com dívidas técnicas, que é de 1.31 trilhões e segue crescendo, considerando que em 2018 era 14% menor. Outrossim, a má qualidade de código também inclui a falta de testes, como integração, *end-to-end*, testes unitários, testes de sobrecarga, entre outros. Isso é comentado desde muito antes, como mostra Tassey (2002), em que fora estimado um custo com testes de software entre 22.2 e 59.5 bilhões de dólares.

De fato, é de se observar que o problema acontecia já em 2002, mesmo com grandes obras literárias sobre o tema já existindo naquela época. Martin Fowler já tinha lançado diversos títulos como o Refactoring[1] (1999) em que é discutida a importância das refatorações e dos testes. A obra Extreme Programming[2] (1999) já existia com Kent Beck trazendo ideias de como desenvolver um *software* melhor e com um estilo de desenvolvimento bem diferente. A obra The Pragmatic Programmer já tinha sido lançada por Andy Hunt e Dave Thomas, que traziam um pensamento crítico aos programadores.

Além dessas importantes obras, o Manifesto do Desenvolvimento Ágil de Software já tinha sido publicado, trazendo uma série de práticas e valores para desburocratizar o desenvolvimento de software. Infelizmente, segundo Thomas (2014) em sua palestra “Agile is Dead”, os valores foram perdidos, pois, com o tempo as ideias originais do manifesto foram deturpadas e os conceitos esquecidos, mascarando diversas ferramentas burocráticas com a palavra ágil.

De acordo com uma pesquisa realizada pelos autores deste trabalho, a maioria dos programadores de parte da região noroeste e microrregião de Jales ainda trabalham com sistemas legados. Ademais, a maior parte dos desenvolvedores prefere trabalhar em empresas

[1] FOWLER, M. **Refactoring**: Improving the Design of Existing Code. New York: Addison-Wesley Professional, 1999.

[2] BECK, K.; ANDRES, C. **Extreme programming explained**: embrace change. New York: Addison-Wesley Professional, 1999.

visando iniciar novos projetos, evitando código legado. Isso pode causar uma perda de talentos por parte da empresa, além da dificuldade de manutenção e o custo que isso traz a longo prazo.

Outra grande parte dos desenvolvedores reclamaram de começarem alguma tarefa e logo em seguida ser pausado porque não tinha mais prioridade, ou o cliente muitas vezes nem existia ou nem era aquilo que ele realmente queria, um grande problema de gerenciamento do projeto e experiência do usuário. Frente ao exposto, o objetivo desse trabalho é entender parte desse mercado.

Assim, a Seção 2 apresenta os principais conceitos envolvidos no desenvolvimento de software. A Seção 3 apresenta as metodologias usadas no desenvolvimento deste trabalho. Na Seção 4, são apresentados os resultados e discussões da pesquisa realizada. Finalmente, na Seção 5, é apresentada a conclusão.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentados os principais conceitos relacionados a ES que são importantes para este trabalho.

2.1 PADRÕES DE PROJETOS

No inglês existem três palavras principais usadas na área de desenvolvimento de *sofware* que se traduzem para padrão, sendo *default*, *standard* e *pattern*. Entretanto, cada uma possui um significado distinto, e entender isso é de suma importância para falar sobre padrões de projetos.

A palavra *default* se trata de uma opção pré-selecionada quando nenhuma alternativa foi escolhida. Segundo o dicionário Cambridge (CAMBRIDGE UNIVERSITY, 2022), “é aquilo que existe e acontece se nenhuma alteração for feita”. O termo *standard* está mais próximo de uma norma com requisitos para serem seguidos, tendo como definição um nível de qualidade e está comparável com critério ou parâmetro. E o termo *patterns* se trata de um modelo recorrente de comportamento, que não está relacionado ao certo ou errado, mas sim apenas a algo que acontece recorrentemente.

Logo, padrões de projeto é um conjunto de *patterns*, ou seja, comportamentos recorrentes que aconteceram e continuam acontecendo em projetos relacionados ao código e que não é regra para o bom desenvolvimento. Assim, alguns livros que se tornaram referências foram lançados para catalogar eles como o "Design Pattern[3]" (1994), "Enterprise Integration Patterns[4]" (2003), entre outros. Ter conhecimento dos mais conhecidos ajuda a compreender melhor o código de diversos sistemas que provavelmente aplicaram em algum momento determinadas estratégias.

Portanto, com eles é possível ter um direcionamento que ajuda na hora de desenvolver boas soluções, ou até mesmo se comunicar com mais facilidade entre a equipe técnica.

2.2 DESENVOLVIMENTO ÁGIL DE SOFTWARE

O manifesto para desenvolvimento ágil de software foi um documento criado em 2001 pelos autores mais importantes quando se trata de desenvolvimento de software, com o objetivo de propor as melhores práticas de desenvolvimento. Assim, por trás do manifesto, há doze princípios que tiveram como foco de sua origem a desburocratização dos processos dá época.

---

[3] GAMMA, E. *et al*. **Design patterns**: elements of reusable object-oriented software. New York: Addison-Wesley Professional, 1994.

[4] HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns**: Designing, Building, and Deploying Messaging Solutions. New York: Addison-Wesley Professional, 2003.

Dentre esses princípios, o mais importante para esse trabalho é: a prioridade de satisfazer o cliente através da entrega contínua e adiantada de *software*. Os processos ágeis se adequam as mudanças para que tenham vantagens competitivas, então deve sempre se aceitar mudanças de requisitos. Times que são auto-organizáveis tendem a ter as melhores arquiteturas, requisitos e *designs*.

2.3 DESENVOLVIMENTO LIMPO

Segundo Martin (2009), a ideia de que um código bem desenvolvido é a premissa mais robustas dentre todas na área, pois o desenvolvimento de um código ruim pode fazer com que inúmeros problemas comecem a surgir nas organizações. Quanto mais fácil de se entender algo que foi escrito, mais fácil é também a manutenção e alteração, que pode ser feita sem que se leve muito tempo.

Quando se trata de custos, deve-se considerar o tempo em que os desenvolvedores perdem apenas tentando entender o código que está sem manutenção há anos ou simplesmente foi desenvolvido de qualquer maneira.

Sendo assim, o conceito de código limpo pode ser definido como uma maneira de tornar o desenvolvimento e a manutenção cada vez mais simples. Um sistema nunca está totalmente finalizado, afinal o código envelhece e pode se tornar obsoleto.

Com essas informações, fica evidente a necessidade de se ter princípios para um desenvolvimento limpo, algo que vai muito além da boa arquitetura e boas práticas computacionais, é justamente essa importância que deve ser dada ao código limpo.

2.4 EXPERIÊNCIA DO USUÁRIO

Na década de 1990, Donald Norman deu origem ao termo Experiência do Usuário (do inglês User Experience), utilizando as próprias palavras: “é tudo o que se refere à sua experiência com um produto, mesmo que você nem esteja perto dele”.

Com o tempo, o conceito foi evoluindo e sendo utilizado de maneira superficial ao que realmente se refere. Segundo Norman (2016), “atualmente o termo é terrivelmente mal utilizado, por pessoas que falam sobre a criação de website e aplicativos, quando na verdade ele é tudo, pode ser desde a conversa sobre o assunto, até o último contato que temos com ele”.

Segundo Norman (2016), existem três níveis de experiência do usuário. O primeiro nível é chamado de interação única, e é responsável pela realização de tarefas onde o usuário não deverá ter problemas para identificar passos simples. O nível de jornada é responsável por um escopo maior buscando atingir uma meta. O último nível é o de relacionamento, que envolve todas as interações existentes que conectam os usuários à empresa.

Logo, pode-se dizer que a experiência do usuário se refere às percepções dele diante de todas as interações que ele tem com o produto. A importância do UX é fundamental para uma empresa, já que é uma forma de garantir que seus usuários se sintam bem utilizando o produto e permaneçam mais tempo.

2.5 TESTES DE SOFTWARE

Tom Kilburn é considerado uma das primeiras pessoas que desenvolveram um software. O sistema dele foi lançado em 1948 na Universidade de Manchester para realizar cálculos matemáticos utilizando instruções de máquina. Assim, desde essa época, é necessário testar para saber se realmente estava funcionando da forma adequada. Entretanto, os testes eram, e infelizmente ainda são feitos por muitas pessoas, por meio do *debug* (depurador) ou executando e manualmente verificando se está funcionando e se as saídas correspondem às entradas.

Felizmente, na década de 1980, os times começaram a ter uma visão mais ampla sobre testes. Assim, eles começaram a integrar o processo de desenvolvimento de *software*, a fim de melhorar o nível de qualidade. Assim, de acordo com Yaroshko (2018), em 1990, os testes chegaram a um nível mais alto, e fez surgir o desenvolvimento de metodologias e ferramentas novas e poderosas para gerenciar os processos de teste e automação desses processos.

Atualmente, há diversos tipos de testes, como, por exemplo, testes de sobrecarga, testes *end-to-end*, testes de regressão, testes de integração, testes de performance, etc. Além de diversas ferramentas de *DevOps*[5]. O objetivo é garantir que não haverá problemas antes do cliente ter acesso a aplicação, e fazer isso de forma automatizada.

2.6 SISTEMAS LEGADOS

Segundo Bennett (1995), sistemas legados podem ser definidos como grandes sistemas de *software*, com os quais os membros da equipe não sabem lidar, mas que são vitais para a organização.

Ademais, isso pode ir além do *software*, atingindo hardware e processos de gestão em si, pois, em um ecossistema legado, normalmente, há contratos externos que ainda precisam ser mantidos, infraestruturas antigas e complexas de mudar, além de diversos problemas em torno do tema.

2.7 DÍVIDAS TÉCNICAS

O termo originalmente foi usado por Ward Cunningham quando precisou explicar para seu chefe os problemas da não refatoração de um código que ele tinha criado, então ele fez uma analogia as dívidas do mercado financeiro para mostrar as consequências disso.

Segundo Holvitie et al. (2018), a dívida técnica descreve as consequências das ações de desenvolvimento de *software* que priorizam intencionalmente ou não o valor do cliente e/ou as restrições do projeto. Entre essas restrições, pode-se destacar prazos de entrega, questões sobre implementação mais técnica e considerações de *design*.

As principais consequências das dívidas técnicas são: a) redução da agilidade — testes serão mais lentos e modelagem de domínio e comutação mais dolorosas; b) impactos financeiros — causando problemas com o orçamento. Logo, combater esses problemas é importante para saúde da empresa, entretanto, é importante saber que dívida técnica não é sinônimo de desorganização. Segundo Martin (2009), se trata de decisões estratégicas tomadas com base em restrições reais, e que deverá ser paga futuramente, e não algo irracional consequência de falta de *know-how* e profissionalismo.

2.8 CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE

Quase todos as áreas têm um padrão definido para oferecer orientação das melhores práticas e formas de se fazer algo. No caso da área de software, os desenvolvedores têm o SDLC, que estabelece modelos para que sigam durante o desenvolvimento e cada modelo possui um conjunto de etapas que os engenheiros de software seguem em um projeto de desenvolvimento.

[5] O DevOps é a combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura (AWS, 2022).

Segundo Sommervile (2011), existem quatro áreas básicas do ciclo de vida do desenvolvimento de software, sendo elas: especificação, desenvolvimento, validação e evolução, e são organizadas de forma diferente conforme o processo. Por exemplo, no modelo em cascata são organizadas em sequência, enquanto no desenvolvimento incremental são intercaladas. Assim, a maneira como essas atividades serão feitas depende do tipo de software, das pessoas e das estruturas organizacionais envolvidas.

Nos dias atuais, as empresas trabalham em um ambiente global, competitivo e que necessita de mudanças rápidas, o que faz com que a qualidade seja sacrificada para que ser entregue. Ainda nas palavras de Sommervile (2011) “[...] muitas empresas estão dispostas a trocar a qualidade e o compromisso com requisitos do software por uma implantação mais rápida do software de que necessitam”.

Assim, o modelo de desenvolvimento ágil de software acaba sendo, atualmente, o mais utilizado, pois ele busca uma flexibilidade e agilidade maior em comparação aos tradicionais que tendem a ser mais rígidos e controlados.

3 METODOLOGIA

Para o desenvolvimento do presente trabalho, inicialmente foi conduzido um levantamento bibliográfico em artigos, livros, sites oficiais de empresas, etc, o qual teve como objetivo, compreender os principais conceitos em torno do tema “qualidade do código”.

Leonhard Euler criou uma das equações considerada, por muitos estudos, a mais bela da matemática, equação de Euler, em que sua beleza foi comparada a da Mona Lisa, de Leonardo Da Vinci, a de David, de Michelangelo e ao soneto, de Shakespeare “captura a mesmíssima essência do amor”, pois, mesmo sendo extremamente simples, ela consegue unir os elementos mais famosos da matemática e se tornar uma das mais importantes equações criadas. E isso também é um dos pilares que faz um código ser considerado bom, sua simplicidade e expressividade.

Além disso, um bom código deve também ser de fácil manutenção, reaproveitável, ter testes e entregar valor ao seu usuário, como mostra no código limpo. Ademais, a equipe deve estar alinhada, seguindo arquiteturas definidas e pagando as dívidas técnicas que são geradas com o tempo.

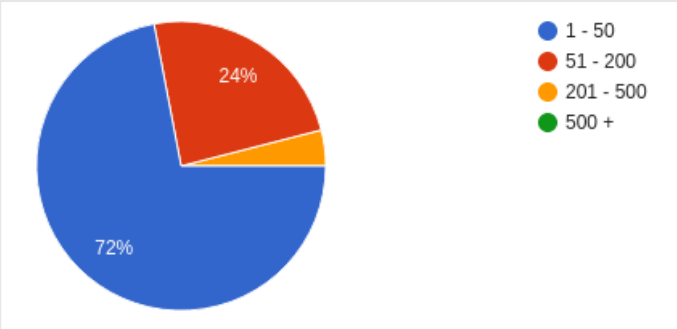
Deste modo, para entender como as empresas de desenvolvimento de *software* da região noroeste do estado de São Paulo lidam com o tema qualidade do código, foi conduzida uma pesquisa on-line entre os dias 14/04/2022 e 03/05/2022 com auxílio da ferramenta Google Forms.

Dentre as principais perguntas, pode-se destacar “A empresa que você trabalha tem a cultura de ter testes de código?”, “Possui uma arquitetura definida de código nos projetos que você trabalha?” e “O pagamento das dívidas técnicas é um processo contínuo da empresa em que você trabalha?”, com as quais foi possível constatar diversos aspectos negativos sobre como as empresas lidam com a qualidade do código. Na Seção 4, há o detalhamento e a discussão dessa pesquisa.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta seção são apresentadas as análises dos dados da pesquisa citada na Seção 3, que obteve 25 respostas. Vale ressaltar que a pesquisa foi feita com o objetivo de entender os desenvolvedores do noroeste paulista, principalmente na região de Jales, estado de São Paulo. Assim, as respostas de fora dessa região foram desconsideradas, assim como respostas em que a cidade ficou em branco. Ademais, a maior parte das empresas, como mostrado no Gráfico 1, são empresas de médio e grande porte.

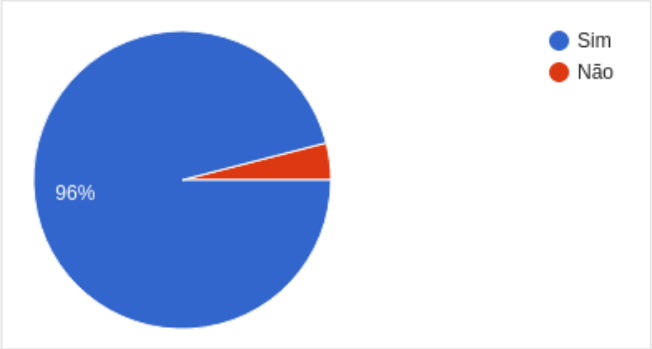
Gráfico 1 – O porte das empresas em número de funcionários



Fonte: Elaborado pelos autores.

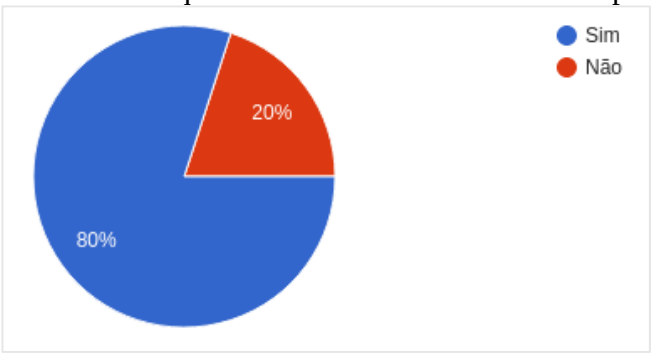
De acordo com o Gráfico 2, 96% dos desenvolvedores citaram que a empresa em que trabalham possuem sistemas legados. Analisando o Gráfico 3, é possível constatar que 80% dos entrevistados levariam isso em conta como um ponto negativo na hora de trocar de empresa. Além disso, 92% acreditam que sistemas legados afetam negativamente na produtividade como mostra no Gráfico 4.

Gráfico 2 – A empresa possui sistemas legados?



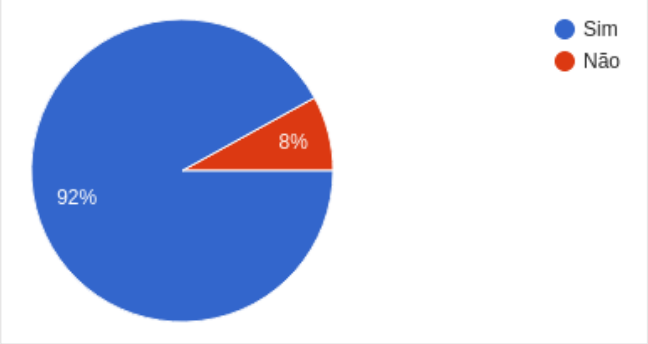
Fonte: Elaborado pelos autores.

Gráfico 3 – Trabalhar com sistemas legados é um fator que faria o colaborador sair da empresa?



Fonte: Elaborado pelos autores.

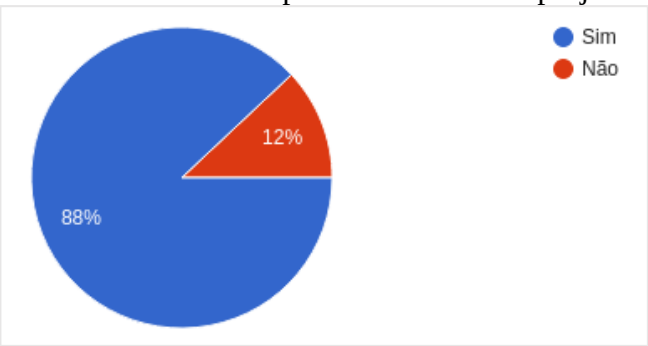
Gráfico 4 – Trabalhar com sistemas legados impacta negativamente na produtividade?



Fonte: Elaborado pelos autores.

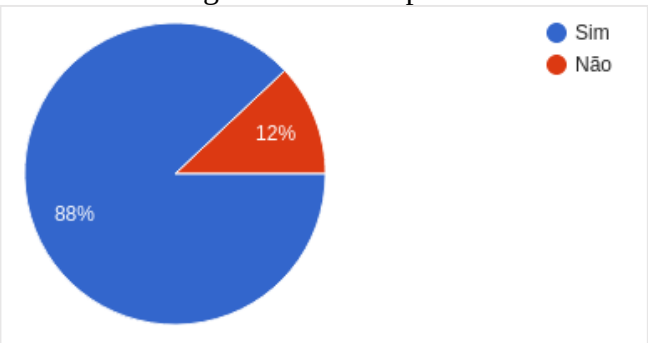
Outro problema evidenciado no Gráfico 5 é sobre mudanças de requisitos que aconteceram em 88% das empresas. Nesse sentido, algumas empresas, devido à falta de planejamento ou mão de obra, acabam constantemente interrompendo projetos e alocando os desenvolvedores desses projetos em outros que acreditam ser mais relevantes, com o objetivo de atingir metas nos prazos definidos. Assim, diversos projetos são abandonados, prejudicando a produtividade dos realocados, como é mostrado no Gráfico 6.

Gráfico 5 – Já aconteceu mudança de requisitos de forma repentina durante um projeto?



Fonte: Elaborado pelos autores.

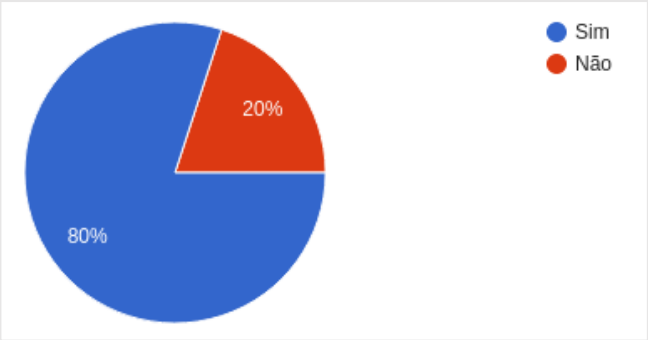
Gráfico 6 – As mudanças de requisitos repentina impacta negativamente na produtividade?



Fonte: Elaborado pelos autores.

Outrossim, 80% dos entrevistados acreditam que essa preocupação com prazos acaba prejudicando a qualidade do código desenvolvido, como é referenciado pelo Gráfico 7 da pesquisa.

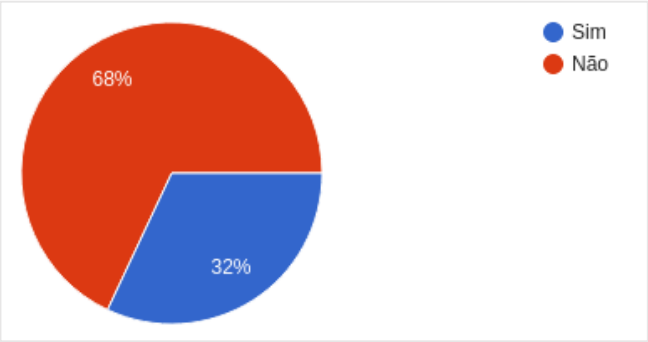
Gráfico 7 – Os prazos interferem na qualidade de código?



Fonte: Elaborado pelos autores.

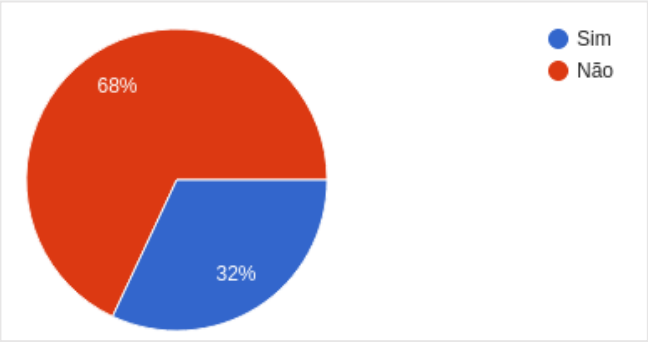
Segundo o Gráfico 8, mais da metade das empresas não possuem nenhum tipo de auditoria em relação a qualidade de código e nem treinamento relacionados as boas práticas de programação que são fornecidas em concordância com o Gráfico 9. Ademais, quase da metade das empresas não possui um departamento de QA (*Quality Assurance*), como exposto no Gráfico 10, e 64% não possuem uma cultura de testes de código, segundo o Gráfico 11.

Gráfico 8 – As empresas possuem auditoria em relação a qualidade do código?



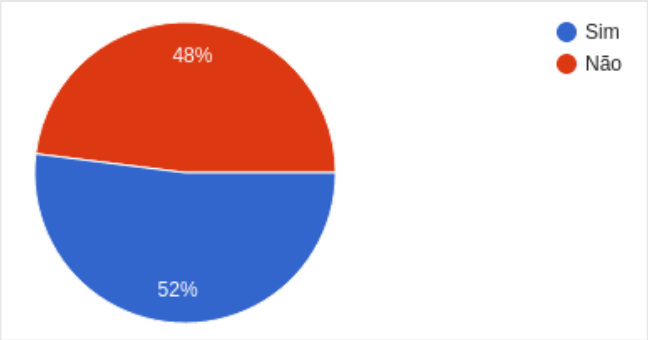
Fonte: Elaborado pelos autores.

Gráfico 9 – Existe treinamento nas empresas em relação as boas práticas de programação?



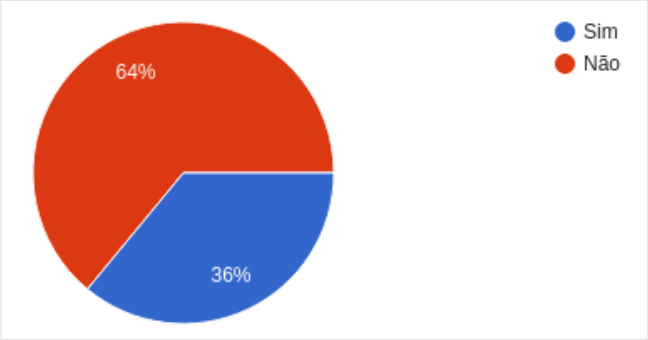
Fonte: Elaborado pelos autores.

Gráfico 10 – Existe departamentos de QA nas empresas?



Fonte: Elaborado pelos autores.

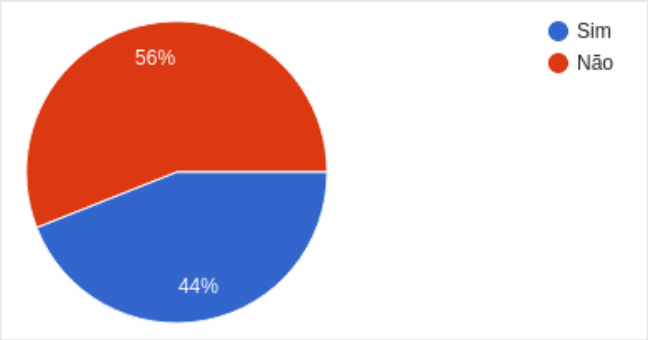
Gráfico 11 – Existe cultura de testes nas empresas?



Fonte: Elaborado pelos autores.

Outro problema comum é não existir em todas as empresas uma arquitetura de código definida. Segundo o Gráfico 12, quase 60% das empresas não têm um padrão para seguir, como MVC (*Model View Controller*), MVP (*Model View Presenter*), MVVM (*Model-View View-Model*) etc.

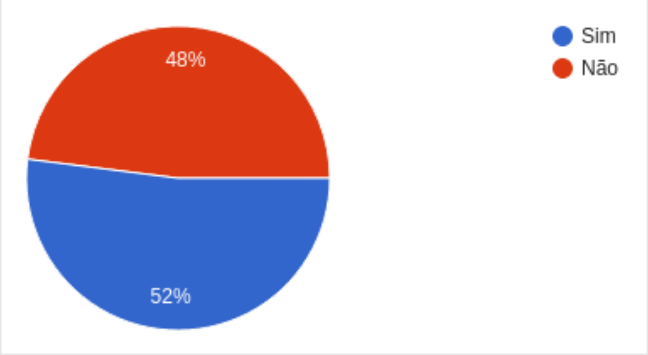
Gráfico 12 – Existe arquiteturas definidas nos projetos das empresas?



Fonte: Elaborado pelos autores.

De acordo com o Gráfico 13, mesmo com diversos problemas a maior parte das empresas não possuem um processo de pagamento de dívidas técnicas de forma contínua.

Gráfico 13 – Existe um pagamento contínuo das dívidas técnicas nas empresas?



Fonte: Elaborado pelos autores.

5 CONSIDERAÇÕES FINAIS

Como analisado por meio da pesquisa, é fato que existem falhas nos processos relacionados ao SDLC. A grande maioria das empresas não possui um departamento de QA ou uma cultura de testes no geral, o que, segundo a literatura especializada (Seção 2), é uma das bases que garante a qualidade do software.

Além disso, mesmo as que tentam seguir os processos devem tomar cuidado, pois escolher um modelo é algo complexo. Ademais, para sistemas legados, que é o cenário da maioria, o modelo ágil traz uma série de problemas, pois sacrifica a qualidade ao encurtar os prazos, a fim de entregar de forma rápida para o cliente.

Além disso, é praticamente impossível estimar com precisão os requisitos no início do projeto quando se segue o modelo de desenvolvimento ágil, o que faz com que eles mudem repentinamente na maioria das vezes. De acordo com Sommervile (2011), não será possível ter um conjunto de requisitos estáveis, e muitas vezes só será claro o que era preciso, depois da entrega do sistema e à medida em que os usuários ganharem experiência.

Entretanto, segundo o que foi apresentado na Seção 4, é problemático ter um cenário com prazos curtos e mudança de requisitos repentina, pois isso impacta na produtividade do desenvolvedor e na própria qualidade. Outrossim, isso acaba com todas as possibilidades de pagamento contínuo das dívidas técnicas, e trabalhar com sistemas legados é necessário que seja feito exatamente o oposto.

Segundo McAnallen (2022), para a implementação prática de métodos ágeis em um ambiente de sistema legado, seria melhor adotar um modelo híbrido, pois é mais vantajoso do que apenas um modelo específico.

Logo, é necessário que seja analisado o contexto empresarial e encontre um equilíbrio entre a qualidade e a entrega. Não obstante, fornecer o treinamento e reconhecimento necessário para os colaboradores devem ser a base para que isso funcione.

REFERÊNCIAS

AWS. **O que é o DevOps?** Disponível em: https://aws.amazon.com/pt/devops/what-is-devops/. Acesso em: 20 maio 2022.

BENNETT, K. Legacy systems: coping with success. **IEEE Software**, v. 12, n. 1, jan. 1995. Disponível em: https://ieeexplore.ieee.org/document/363157. Acesso em: 15 jan. 2022.

CAMBRIDGE UNIVERSITY. **Cambridge dictionary**: default. Disponível em: https://dictionary.cambridge.org/pt/dicionario/ingles-portugues/default?q=default+. Acesso em: 10 mar. 2022.

HOLVITIE, J. *et al*. Information and software technology: technical debt and agile software development practices and processes: an industry practitioner survey. **Information and Software Technology**, v. 96, p. 141-160, abr. 2018. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584917305098?via%3Dihub. Acesso em: 10 jan. 2022.

KRASNER, H. The cost of poor software quality in the US: a 2020 report. **CISQ Consortium for Information & Software Quality**, p. 1-46, jan. 2021. Disponível em: https://www.it-cisq.org/pdf/CPSQ-2020-report.pdf. Acesso em: 10 mar. 2022.

MARTIN, R. C. **Código limpo**: habilidades práticas do Agile software. Rio de Janeiro: Alta Books, 2009.

MCANALLEN, M. *et al*. **Tailoring extreme programming for legacy systems**: lessons learned. Disponível em: https://ulir.ul.ie/bitstream/handle/10344/2617/McAnallen.pdf;jsessionid=3798CCCC05ACBD34C391683742D7A1A8?sequence=2. Acesso em: 15 maio 2022.

NORMAN, D. **O termo UX**. [*S.l.*]: NNgroup, 2016. 1 vídeo (1:49min.). Publicado por NNGroup. Disponível em: https://www.youtube.com/watch?v=9BdtGjoIN4E. Acesso em: 5 mar. 2022.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. 8. ed. São Paulo: AMGH, 2021.

SOMMERVILE, I. **Engenharia de software**. 9. ed. São Paulo: Pearson Education, 2011.

STRIPE. **The developer coefficient**: developers have the ability to raise global GDP by $3 trillion over the next 10 years. 2018. Disponível em: https://d37ugbyn3rpeym.cloudfront.net/newsroom/the-developer-coefficient.pdf. Acesso em: 10 mar. 2022.

TASSEY, G. **The econimoc impacts of inadequate infrastructure for software testing**: final report. Washington, DC: NIST, 2002. Disponível em: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf. Acesso em: 10 mar. 2022.

THOMAS, D. **The death of agile**. Dallas: Thoughtworks, 2014. 1 vídeo (47:45min.). Disponível em: https://www.thoughtworks.com/talks/the-death-of-agile. Acesso em: 15 jan. 2022.

YAROSHKO, A. **Small history of software testing**. 2018. Disponível em: https://www.utest.com/articles/small-history-of-software-testing. Acesso em: 15 jan. 2022.

AGRADECIMENTOS

Agradecemos a Deus, nossas famílias, ao orientador deste trabalho Jorge Gregório, a todos professores, em especial o querido professor de inglês Carlos Alberto, e a todos os colegas que passaram o tempo com a gente durante os 3 anos de curso.