Implantação da prática de *code review* em um modelo de desenvolvimento de software: um estudo de caso

Vinicius Junqueira Schettino

JUIZ DE FORA

JUNHO, 2017

Implantação da prática de *code review* em um modelo de desenvolvimento de software: um estudo de caso

Vinicius Junqueira Schettino

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento da Ciência de Computação
Bacharelado em Sistemas de Informação

Orientador: Marco Antônio Pereira Araújo

JUIZ DE FORA
JUNHO, 2017

Implantação da prática de *code review* em um modelo de desenvolvimento de software: um estudo de caso

Vinicius Junqueira Schettino

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Marco Antônio Pereira Araújo
Doutor

José Maria Nazar David
Doutor

Regina Maria Maciel Braga Villela
Doutor

JUIZ DE FORA

30 DE JUNHO, 2017

Resumo

A qualidade estrutural de software e a aderência à padrões de codificação estão entre os principais desafios do desenvolvimento atual. Tais conquistas são históricamente relacionadas ao decréscimo de defeitos funcionais de software e à capacidade de equipes de desenvolvimento para responder eficientemente a mudanças de requisitos e definições estratégicas, situações comuns no cenário de produção de software atual. A prática de *Code Review* é tida como umas das principais ferramentas para apoiar tais conquistas, ao acarretar embate de ideias, disseminação de conhecimento e a elaboração de soluções melhores. Por ser uma tarefa necessariamente humana, os custos associados não são desprezíveis e há possibilidade de conflitos e desentendimentos entre membros das equipes envolvidas. Assim, a implantação desta etapa no processo de desenvolvimento deve ser planejada levando em consideração tais fatores. Neste trabalho, busca-se avaliar tais asserções em um ambiente de desenvolvimento real. Da reengenharia dos processos vigentes à escolha das ferramentas e das *checklists* utilizadas, passando pelo plano de extração e análise de métricas, realizou-se a inserção da prática na indústria e observou-se a alteração de indicadores-chave relacionados à qualidade estrutural de software. Os dados obtidos se mostraram convergentes com a literatura indicando, entre outros resultados, redução de até 31% na duplicação de código e até cerca de 71% de queda da dívida técnica. Também foi possível observar que 75% desenvolvedores participantes perceberam que as revisões habitualmente nutriram aprendizado e a implementação de soluções melhores.

Palavras-chave: *Code Review*, Qualidade de Código, Reengenharia de Processos de Software

Abstract

Internal software quality and adherence to coding patterns are among the main challanges of current software development. Such conquers are historically bounded to the decrease of functional defects and the team's ability of efficiently respond to requirements and planning changes, which are common situations on the modern software production scenario. The *Code Review* is considerated one of the most resorted techniques to support such conquers, leading to discussions, knowledge spread and elaboration of better solutions. As this is a essentially a human task, the associated costs are not negligible, and it is possible to happen conflicts or strifes between the team members involved. Thus, this step on the development process must be planned taking into account such aspects. From the process reengineering to screening tools and the assamble of the checklists needed, passing by the metrics extraction and analysis plan, this pratice was grafted on the industry and sought for changes on key indicators related to internal software quality. The results converge with the literature pointing to, among other metrics, to decrease of code duplication over 31%, as 71% decay on tecnical debt. It was noticed that 75% of the developers perceived the revisions usually fostered knowledge and the implementation of better solutions.

Keywords: Code Review, Software Quality, Software process reengineering

Agradecimentos

Demasiado prolixo seria eu caso tentasse citar todos aqueles que, de alguma forma, contribuiram para a elaboração deste trabalho. Tentá-lo-ei de qualquer forma, sabendo do improvável sucesso e já aproveitando para me desculpar com aquele ou aquela que se sentir exclúido desta seção.

Primeiramente cito minha família, em especial meus pais Geórgia e Bruno, pela paciência pelas horas trancado no quarto e da ausência de compromissos familiares. Este suporte foi fundamental para execução desta pesquisa e nunca poderá ser devidamente retribuído. Registro também minha namorada Mila, à qual sou eternamente grato pela resistência nas conversas sem fim sobre assuntos (para ela) desinteressantes e na revisão das longas páginas aqui reproduzidas.

Incluo todos os professores que tive, que contribuiram para este trabalho com seus conhecimentos e fundamentaram minha vontade de aprender e meu gosto pela ciência.

Não posso esquecer dos meus colegas de trabalho nem das organizações que trabalhei, as quais desafiaram e estenderam as soluções propostas, possibilitando espaço para arriscar e melhorar cada vez mais.

Por fim um agradecimento especial ao povo brasileiro, que financiou indiretamente este trabalho. Gosto de pensar que esta pesquisa irá, mesmo em sua patente modéstia, servir de algo para o desenvolvimento dessa grande nação.

"Não me peça que lhe faça uma canção como se deve // Correta, branca, suave, muito limpa, muito leve // Sons, palavras, são navalhas e eu não posso cantar como convém // Sem querer ferir ninguém"

Belchior (Eu sou apenas um rapaz latino americano)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
PICOC	*Population, Intervention, Comparison, Outcome, Context*
GQM	*Goal, Question, Metric*
MCR	*Modern Code Review*
QA	*Quality Assurance*
LOC	*Lines of Code*
IDE	*Integrated Development Environment*
API	*Application Programming Interface*
SQALE	*Software Quality Assessment based on Lifecycle Expectations*

1 Introdução

1.1 Apresentação do tema

O *code review* é um processo difundido na cultura de organizações de diversos portes e áreas de atuação. A técnica constitui da nomeação de um desenvolvedor para revisar uma modficação do código fonte submetido por outro desenvolvedor, tendo como base um conjunto de diretrizes e padrões a serem observados. A prática adquire formatos e particularidades de acordo com o contexto e objetivos aos quais é associada. Contudo, a melhoria da qualidade do código e a disseminação do conhecimento entre os diversos membros da equipe são objetivos geralmente vislumbrados com a aplicação da prática.

A qualidade do código pode ser inferida pela inspeção, entre outras métricas, da dívida técnica e mau cheiros de código (ou *code smells*). O primeiro é uma metáfora para representar a necessidade de refatoração e atenção à fatores não funcionais de projetos de software (KRUCHTEN; NORD; OZKAYA, 2012), enquanto o segundo representa trechos de código com alto risco de proporcionar defeitos e detentores de baixa manutenibilidade (FOWLER et al., 1999). A revisão de código pode fomentar a discussão ténica sobre as soluções propostas e levar a implementações e modelos melhores, impactando diretamente em tais métricas.

1.2 Problema

Durante o ciclo de desenvolvimento, é comum que haja pouco contato entre os desenvolvedores e o código em produção por outros colegas. Este cenário não favorece a disseminação do conhecimento, aderência a padrões e pode aumentar a ocorrência de duplicação de código. Muitas vezes esta segregação inibe o debate que poderia levar à implementação de soluções melhores.

1.3 Objetivos

Neste trabalho foi possível pesquisar, modelar e implantar técnicas de *code review* no modelo de desenvolvimento de uma empresa de médio porte do ramo de desenvolvimento de hardware e software. Propõe-se, por intermédio de uma plataforma de inspeção de código, inferir métricas de qualidade, comparando estes indicadores antes e depois da implantação da prática de *code review*. Através da técnica Goal/Question/Metric (GQM) (BASILI; WEISS, 1984), os objetivos deste trabalho são descritos como:

> "Caracterizar **o efeito do** *code review* na qualidade estrutural de software **do ponto de vista dos** desenvolvedores **no contexto de** pequenas equipes de desenvolvimento de software."

1.4 Hipóteses

O presente trabalho está pautado na verificação das seguintes hipóteses

- A prática de revisão de código pode impactar positivamente em indicadores de qualidade do software;
- a prática de revisão de código pode fomentar a disseminação de conhecimento, debates e senso de responsabilidade compartilhada do código fonte.

1.5 Organização do trabalho

Além da introdução, este trabalho é composto de outros 6 capítulos. No capítulo 2 é apresentado o mapeamento sistemático da literatura relacionada. No capítulo 3, relata-se o contexto da organização, enquanto o modelo de desenvolvimento proposto para aplicação da prática de *code review* é elucidado na seção 4, incluindo a escolha das ferramentas consideradas necessárias. No capítulo 5 estão descritas as técnicas utilizadas e a implantação do processo de *code review*. Na seção 6, discorre-se sobre resultados obtidos, enquanto as considerações finais sobre o trabalho se encontram na seção 7.

2 Mapeamento Sistemático da Literatura

2.1 Planejando o mapeamento

Para a revisão da literatura relacionada à este trabalho, escolheu-se uma abordagem sistemática, como descrito por Kitcheham e Charters (2007). Este método auxilia na conjectura de uma base teórica sólida para o trabalho aqui proposto, uma vez que promove uma revisão imparcial e ampla do conhecimento disponível sobre determinado assunto. Além disso, um modelo sistemático, baseado em um protocolo estruturado, facilita futura revisão e reaplicação da pesquisa.

Através do protocolo foi possível definir uma *string* de busca, os critérios de inclusão e exclusão de referências e a diretrizes para eleição das bases de dados a serem utilizadas. O *output* desta busca será então utilizado como fundamentação conceitual para a apresentação da revisão bibliográfica.

2.1.1 Objetivos e *string* de busca

O padrão para busca foi determinado de acordo com a técnica Goal/Question/Metric, (GQM) (BASILI; WEISS, 1984). O objetivo foi delineado como "**analisar** as métricas, modelos e hipóteses utilizadas na implantação da prática de *code review*, **para então** definir a metodologia de trabalho e avaliação **do ponto de vista** de qualidade de software, **no contexto** de desenvolvimento de sistemas."

As palavras-chave para busca foram definidas através da estratégia PICOC, que tem como objetivo delimitar cada um dos componentes da questão que a ser feita, evitando ambiguidades e respostas fora do contexto preterido (PETTICREW; ROBERTS, 2008). O resultado pode ser visto na tabela 2.1.

Utilizando as palavras-chave definidas na tabela 2.1, a tabela 2.2 mostra a *string* de busca que aplicada em cada uma das bases elegidas na subseção 2.1.3

Tabela 2.1: PICOC

PICOC	**Palavra-Chave**
Population	*system*, *software*, *application*
Intervention	*code review*
Control	-
Outcome	*metrics*, *case study*, *impact*
Context	*software development*, *software evolution*, *development*

Tabela 2.2: *String* de busca

(*"software"* OR *"application"* OR *"system"*) AND ("*code review*") AND (*"case study"* OR *"metrics"* OR *"impact"*) AND ("*software development*" OR "*software evolution*" OR "*development*")

2.1.2 Critérios de inclusão e exclusão

Para tratar a saída da busca, critérios de inclusão e exclusão foram definidos, como recomendado por Kitcheham e Charters (2007). Esses foram utilizados para decidir se determinados trabalhos encontrados na busca deveriam ou não ser utilizados como fundamentação deste trabalho. Os critérios adotados estão descritos na tabela 2.3

Através da figura 2.1, é possível observar que o número de publicações cresceu no período consultado, se estabelecendo nos anos de 2014 e 2015. Esse pode ser um índicio do crescimento pesquisas relacionadas nos últimos anos. A figura 2.2 mostra que a maior parte dos artigos buscados veio da base Scopus, seguido pela IEEEXplore e por fim a ScienceDirect. Da mesma forma, através da figura 2.3 é possível observar esta tendência

Tabela 2.3: Critérios de inclusão e exclusão

Tipo	**Descrição**
Inclusão	Estudos que descrevem práticas comuns e resultados geralmente atribuidos à prática de *code review*
Inclusão	Trabalhos que relatam estudos de caso ou implantação da prática de *code review*
Exclusão	Livros, Capítulos e outros tipos de material secundário de educação.
Exclusão	Trabalhos que não estejam plenamente disponíveis
Exclusão	Estudos que não contribuem para os objetivos deste trabalho
Exclusão	Trabalhos anteriores ao ano de 2007
Exclusão	Trabalhos que não estejam em inglês

entre os artigos aceitos durante o processo de seleção.

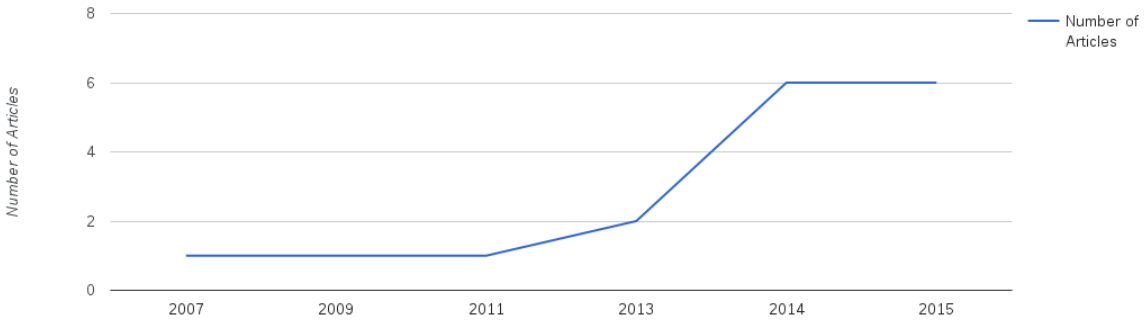

Figura 2.1: Quantidade de publicações na área por ano

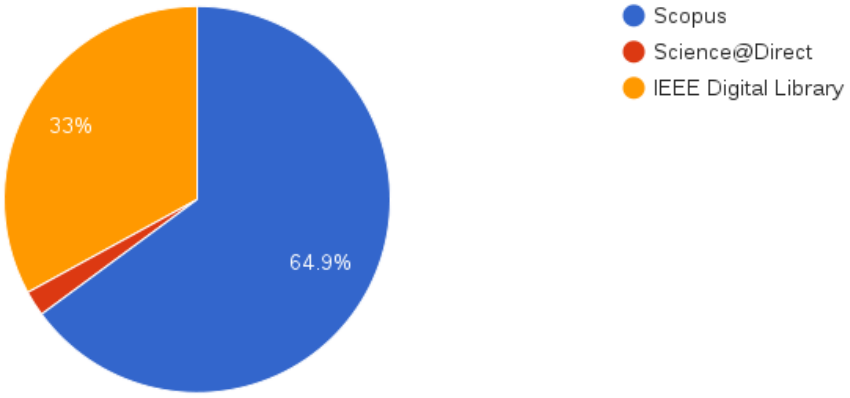

Figura 2.2: Trabalhos encontrados por base

A exclusão de trabalhos anteriores ao ano de 2007 auxiliou no destaque de trabalhos mais atualizados e que já aproveitavam ou atualizavam os resultados e conclusões de trabalhos mais antigos. Por motivo semelhante excluímos livros e capítulos e outros tipos de material secundário de educação.

2.1.3 Escolha das bases de dados

Os seguintes critérios foram utilizados para escolher as bases de dados utilizadas na pesquisa:

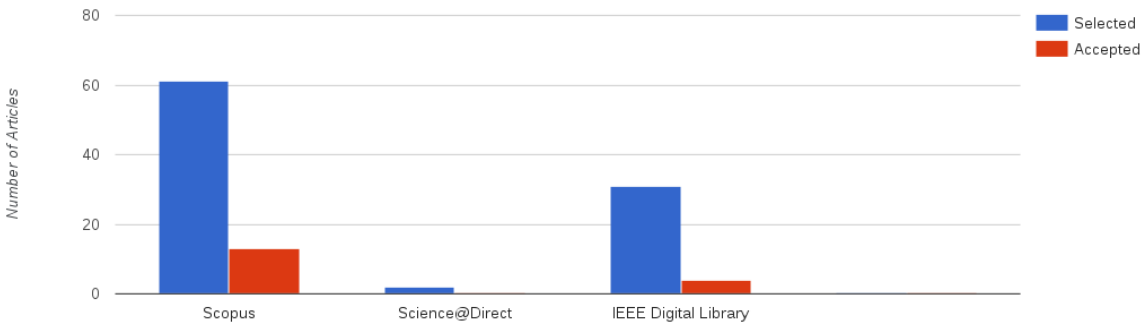

Figura 2.3: Proporção de trabalhos aceitos e rejeitados por base

- Englobar material sobre a área de interesse deste trabalho, Ciência da Computação;
- permitir a busca em partes específicas dos trabalhos, como *abstract* ou título;
- permitir a utilização de expressões lógicas na busca, sem limitação de quantidade de termos ou letras.

As bases de dados Scopus[1], ScienceDirect[2] e IEEExplore[3], disponíveis através do portal CAPES acessível para alunos da UFJF, foram eleitas por atenderem tais diretrizes.

2.2 Condução da revisão

A busca foi executada nas bases definidas na subseção 2.1.3, com pequenas modificações de sintaxe na *string* de busca para adequar ao funcionamento de cada uma delas. Obteve-se 94 resultados, que foram filtrados e organizados com o auxílio da ferramenta Parsifal[4].

Dos artigos selecionados, 20 foram declarados duplicados. Do restante, 52 foram considerados, durante a análise do título e *abstract*, externos ao escopo da pesquisa, com base nos critérios de exclusão apresentados na tabela 2.3. Os 22 restantes foram classificados para leitura completa, dos quais 17 foram considerados relevantes para este trabalho. A análise deste grupo é apresentada na seção 2.3.

[1]https://www.scopus.com/
[2]http://www.sciencedirect.com/
[3]http://ieeexplore.ieee.org/
[4]http://parsif.al

Além dos trabalhos encontrados durante a revisão sistemática, serão adicionados algumas referências clássicas sobre *code review*. Estas, por serem anteriores ao ano de 2007 não foram encontradas devido aos critérios de exclusão utilizados, mas possuem valor histórico para contextualizar este trabalho.

2.3 Apresentação da literatura relacionada

Os primeiros registros de *code review* remontam ao processo de produção de software dos anos 70. A revisão era apoiada por uma série de artefatos criados para boa adaptação com o modelo de desenvolvimento em cascata, como reuniões de revisão, *checklists* de qualidade e auditorias (FAGAN, 1976). Apesar do impacto positivo desta prática na qualidade dos softwares (ACKERMAN; FOWLER; EBENAU, 1984), discutiu-se sobre o custo/benefício da prática e se não deveria ser empregada em aplicações específicas, como sistemas críticos (VOTTA; LAWRENCE, 1993; AURUM; PETERSSON; WOHLIN, 2002).

O surgimento das metodologias ágeis de desenvolvimento e o aumento da preocupação com a qualidade do software contribuiram, entre outros fatores, para a modernização da técnica. Foi averiguada a importância do emprego de ferramentas de apoio, como o Gerrit[5], da flexibilização do processo e da participação e interação dos desenvolvedores na revisão (BACCHELLI; BIRD, 2013).

Este conjunto de práticas é chamado de *Modern Code Review* (MCR), e diversos autores, através dos trabalhos apresentados nesta seção, investigaram seus impactos e características.

A prática de *code review* é tida como uma das principais técnicas para diminuição de defeitos em softwares (BOEHM; BASILI, 2001). Pesquisas mostram que cerca de 75% dos defeitos encontrados durante a revisão estão relacionados à manutenibilidade, enquanto apenas 25% refere-se a defeitos funcionais (BELLER et al., 2014).

Do ponto de vista de codificação, *design* (GAMMA et al., 1995), manutenção e refatoração do código (FOWLER et al., 1999), o *code review* é investigado por diversos autores. No âmbito das decisões de *design*, Morales, McIntosh e Khomh (2015) relatam a diminuição da incidência de *anti-patterns* através da participação ativa dos desenvolvedores

[5]https://www.gerritcodereview.com/

e da cobertura do *code review*. Kemerer e Paulk (2009) discorrem sobre a incidência de más práticas de *design* e sobre a influência de fatores como o número de linhas analisadas por revisão e o número de revisores na eficiência do *code review*. Nesse trabalho também são apresentadas evidências que a eficiência de se analisar 800 linhas de código por hora é cerca de 1/3 da revisão de 200 linhas de código. McIntosh et al. (2014), Bavota e Russo (2015) apresentam resultados convergentes, nos quais o *code review* foi fator relevante na melhoria da qualidade dos sistemas analisados.

Impactos do *code review* na incidência de categorias de defeitos específicas também foram investigados. Meneely et al. (2014) discutem que, para detecção de defeitos relacionados a vulnerabilidades, a experiência do revisor em análise de *bugfixes* de segurança é um fator importante, contrariando resultados de experimentos mais gerais divulgados anteriormente.

Em nichos particulares, como o desenvolvimento de sistemas críticos para controle aéreo, há relatos de *workflows* específicos e impactos significativos em qualidade através da adoção do *code review* (BERNHART et al., 2011).

Apesar de ser uma prática frequente tanto na comunidade *OpenSource* quanto na indústria, o *code review* tem um significado particular no primeiro caso. Um dos fundamentos do desenvolvimento sistemas de código aberto é que haverá uma larga população acompanhando de perto as decisões e mudanças do projeto. Esse princípio foi iconicamente resumido por Raymond (2001): “Dado um número suficiente de desenvolvedores e *testers*, quase todo problema será caracterizado rapidamente e a correção será óbvia para alguém [...]. Muitos olhos fazem todos os *bugs* triviais”.

Este aforismo ficou conhecido como o enunciado da Lei de Linus, em homenagem à Linus Torvalds, criador do *kernel* Linux. Segundo esse ponto de vista, diversos requisitos não funcionais de software, como segurança, manutenibilidade, confiabilidade e usabilidade só são alcançáveis na comunidade *OpenSource* através da participação ativa da comunidade.

Nesse âmbito, o *code review* é comumente praticado com apoio de ferramentas ou listas de email (ASUNDI; JAYANT, 2007). Todos os *patches* devem ser revisados por pelo menos um membro do *core*, seja o autor externo ou não ao projeto (BAYSAL et al.,

2012). Contudo, Bosu e Carver (2014) apresentam resultados apontando que *patches* de autores do *core* do projeto são aprovados mais rápido e com menos escrutínio, enquanto os membros da comunidade geral passam por um processo mais lento e datalhista. O autor defende que esse comportamento pode desmotivar a contribuição para o projeto (seja com desenvolvimento ou revisão), ferindo assim a Lei de Linus.

O amplo espectro de projetos, contextos e objetivos aos quais o *code review* é aplicado precede um conjunto igualmente vasto de características que devem ser estudadas, principalmente sobre seu impacto nos resultados obtidos. Baysal et al. (2013) conduzem um experimento mostrando que fatores como reputação e experiência do autor, bem como a região modificada e o tamanho da modificação influenciam significativamente no tempo de aceitação do *patch*. A experiência do revisor, a quantidade de revisores e a interação entre os envolvidos também apresentam impacto positivo no resultado da revisão (KONONENKO et al., 2015).

Apesar dos benefícios da prática, deve-se considerar também os custos significativos que a permeiam. Como atividade fundamentalmente humana, incrementa a necessidade de pessoal e o tempo de entrega, enquanto outros impactos negativos sociais (como inimizades e desentendimentos) que devem ser observados (BACCHELLI; BIRD, 2013).

Por isso, na prática nem sempre é possível revisar de maneira abrangente todo o *changeset*. Aman (2013) apresenta e avalia um método de estimativa de esforço para revisão de um *patch*, com base em fatores como idade do arquivo, número de linhas de código e histórico de falhas. Esta estimativa ajuda na priorização dos arquivos para revisão através da relação entre probabilidade de introdução de defeitos e esforço de análise, sendo relatado como 42% mais eficiente do que a escolha manual através do histórico de correções. Outros fatores são relevantes ao medir o esforço de uma revisão, como a complexidade ciclomática, proporção de comentários e a soma das linhas adicionadas e linhas removidas (MISHRA; SUREKA, 2014).

Analisando o tempo de revisão, o número de observações e o número de interações entre o revisor e o desenvolvedor, Thongtanunam et al. (2015a) mostram que arquivos com risco mais alto de defeito não são melhor analisados, o que acontece com arquivos já com histórico de defeitos. O autor defende que uma revisão mais detalhista desses

arquivos pode ter um impacto positivo nos resultados da prática.

A escolha do revisor é outro aspecto passivo de investigação. Estudos mostram que entre 4% e 30% das escolhas de revisores não é adequada, levando a revisão a durar até 12 dias extras Thongtanunam et al. (2014, 2015b). Nesses mesmos trabalhos é proposto e aprimorado o RevFinder[6], software que se baseia no histórico de revisores dos arquivos que compõe o changeset.Por fim, Xia et al. (2015) apresentam um método que alia as entradas do RevFinder com o reconhecimento textual, conseguindo recomendações mais eficientes.

[6]http://github.com/patanamon/revfinder

3 Contexto de Desenvolvimento da Pesquisa

A literatura sobre *code review* mostra que a prática geralmente está associada a impactos positivos na qualidade de código, principalmente em relação aos requisitos não funcionais (KEMERER; PAULK, 2009; BELLER et al., 2014; MORALES; MCINTOSH; KHOMH, 2015). Contudo, o modelo de desenvolvimento no qual a prática é inserida, bem como detalhes de implementação do *code review* influenciam significativamente nos resultados obtidos (BAYSAL et al., 2013; KONONENKO et al., 2015). Além disso, ao implantar o processo em ambientes reais de desenvolvimento, é importante monitorar fatores como aumento do custo, atraso no tempo de entrega e fatores sociais negativos (BACCHELLI; BIRD, 2013) e, sempre que possível, mitigar sua ocorrência ou efeitos.

Para conceitualizar os riscos envolvidos, definir valores e abordagens aceitáveis e estipular métodos para quantificação dos resultados, é necessário, além da revisão da literatura, o entendimento do contexto ao qual será apresentada a prática de *code review*.

Neste capítulo discorre-se sobre o ambiente de estudo, aspectos culturais e técnicos da organização e premissas, devidamente justificadas, para a implantação do processo. Baseados na convergência entre estas particularidades e a literatura analisada, o modelo de desenvolvimento será apresentado nas próximas seções.

3.1 Cenário atual

Oito desenvolvedores de software participaram do experimento. Esta é uma das equipes de desenvolvimento de uma *startup* de médio porte em ativdade há cerca de 4 anos, apresentando soluções próprias e para terceiros que envolvem hardware e software. O time de desenvolvimento observado emprega metodologias ágeis para desenvolvimento.

Apesar de não haver oficialmente a adoção de uma metodologia de mercado, como o Scrum[7] ou o XP[8], algumas práticas ágeis conhecidas são empregadas no cotidiano, como por exemplo:

[7] https://www.scrumalliance.org
[8] http://www.extremeprogramming.org/

- divisão do escopo (chamado de *backlog*) em sprints com duração entre 2 e 4 semanas com objetivos bem definidos;
- discussão, planejamento e estimativa coletiva das iterações;
- proximidade com os *stakeholders* e flexibilização do escopo em detrimento dos resultados do projeto;
- foco em produção de valor agregado e entregas pequenas, quantificáveis e constantes.

Esse cenário se mostra, a princípio, propício à implantação de técnicas de qualidade coletivas (como o *code review*) e intuitivamente pode dirimir aspectos sociais negativos como desentendimentos entre envolvidos.

A natureza inovativa do modelo de produção de uma *startup* é visível no modelo de desenvolvimento. A escrita de testes automatizados, padronização de código e busca pela qualidade (tanto interna quanto funcional) parecem ser preoupações constantes dos membros da equipe. Tal cultura pode ser favorável à implantação do *code review* uma vez que mitiga a resistência inicial para adoção da prática a partir do momento que os desenvolvedores conhecem os resultados historicamente atribuídos à sua execução.

A equipe é composta de profissionais de Q.A. (*Quality Assurance*), *Frontend*, *Web* e *Mobile*. Entre as principais linguagens utilizdas estão o PHP, Java (Android) e JavaScript (NodeJS). O controle de versão é feito através do Git[9]. A experiência dos desenvolvedores também é heterogênea; existem alguns com apenas alguns meses de perícia, enquanto outros estão no mercado há 7 anos. Do ponto de vista da formação acadêmica, a disparidade é análoga: há alunos dos primeiros períodos de graduação, enquanto outros possuem mestrado.

O time é responsável por cerca de 7 projetos, cada um em fases do ciclo evolutivo diferentes, como desenvolvimento, manutenção e produção assistida. O tamanho dos projetos também varia; estão entre 12.000 e 25.000 linhas de código (LOC).[10]

[9] https://git-scm.com/

[10] Desconsiderando-se comentários, linhas vazias e código de terceiros (*frameworks* e bibliotecas).

3.2 Objetivos e riscos envolvidos

Com atenção aos trabalhos apresentados e discutidos na seção 2.3, elaborou-se, em conjunto à organização, os objetivos esperados através da prática de *code review*. Foram eles:

- **Qualidade do código**: espera-se que haja uma melhoria nos indicadores internos de qualidade, como a manutenibilidade, legibilidade número de defeitos e complexidade ciclomática;
- **difusão do conhecimento**: o revisor e o autor podem aprender durante o processo, através do contato com tecnologias ou mesmo especificidades de regras de negócio desconhecidas, conhecimento de melhores práticas e soluções alternativas para o problema abordado;
- **senso de responsabilidade compartilhado**: o prática pode fomentar a ótica de autoria compartilhada sobre as diversas partes do código, o que pode influenciar no zelo e na solidariedade sobre aquela porção de trabalho durante as próximas entregas;
- **discussão e implantação de boas soluções**: o *code review* pode influenciar o debate e a discussão sobre as soluções viáveis, transmitindo a visão de profissionais com percepções e experiências diferentes. Isso pode levar a decisões melhores no contexto em revisão.

Outros aspectos, possivelmente resultados da implantação do *code review*, foram considerados indesejados e são listadas medidas de mitigação relacionadas:

- **Esforço extra**: a prática necessita de recursos que poderiam ser aplicados em outras tarefas. Apesar de um certo esforço extra ser inerente à prática, foram tomadas medidas no modelo de desenvolvimento para deixar o processo flexível. Valores como a quantidade de revisores e a quantidade de linhas revisadas foram discutidos com a gerência da equipe para definir-se valores apropriados para o contexto. Tais detalhes serão apresentados nas seções 4.1 e 5.1;

- **aumento do tempo de entrega**: Como há uma nova etapa no modelo de desenvolvimento, o tempo entre o levantamento/delegação da atividade e a atribuição do status "pronto" aumenta. A flexibilidade do modelo é importante para que tarefas urgentes possam ser entregues de maneira direta, enquanto o *code review* ainda pode ser executado posteriormente. Da mesma forma que o item anterior, a relação entre velocidade e qualidade foi levada em consideração durante o desenvolvimento do modelo;

- **problemas sociais ou hostilidades**: O autor pode se sentir ofendido ou desencorajado dependendo do teor da revisão, além da própria sensação de avaliação que pode causar o efeito "nós versus eles". Dependendo das condições do ambiente e do autor, o revisor também pode se sentir constrangido de lançar críticas. Durante a implantação do processo foram conduzidos treinamentos e disponibilizados artigos sobre como realizar o *code review* de maneira cordial e efetiva. As principais diretrizes comunicadas são expostas na seção 5.1, que trata sobre a implantação da prática.

No capítulo 5, a investigação dos métodos de averiguação de tais impactos é apresentada com mais detalhes.

3.3 Ambiente disponibilizado para o experimento

Dentre os produtos contemplados no portfólio da organização, um foi elegido para realizar as observações necessárias a esse trabalho. Um resumo das caracterísitcas estruturais dos componentes estudados, antes da implantação do *code review* está disposto na tabela 3.1.

O produto selecionado é o atual detentor da maior atividade de desenvolvimento atual e contém cinco componentes, ou projetos menores: uma plataforma *backend*, duas APIs e dois aplicativos *mobile*. As linguagens de programação envolvidas são, em sua maioria, PHP, Java e JavaScript.

Este produto possui diversas interfaces com aplicações desenvolvidas por terceiros que são possibilitadas através das APIs que o compõe.

A administração é realizada através do projeto *backend*, enquanto a interface com

Tabela 3.1: Características anteriores à implantação do *code review*

	Backend	**API 1**	**API 2**	**APP 1**	**APP 2**
Versão	4.0	4.0	4.0	2.13	1.9
Linguagem Principal	PHP	PHP	PHP	Java	Java
Linhas de Código (LOC)	24.581	2.939	7.334	11.879	12.007
Linhas Duplicadas	6,3%	35,2%	11,4%	8,7%	3.8%
Débito Técnico	8d 3h	5h 3m	2d 2h	5d 6h	4d 5h
Defeitos por mil LOC	19,28	24,50	15,95	29,13	24,65
Complexidade por mil LOC	128,06	176,25	154,35	179,72	152,16

usuários finais é toda realizada através dos aplicativos para plataforma móvel Android. O projeto encontra-se disponível para o público geral e em regime de produção, desde 2014. Ou seja, desde então o produto é alimentado com dados reais de produção e está atrelado à diretrizes de qualidade de serviço e contratos com os clientes.

4 Modelo de Desenvolvimento Proposto

O modelo de desenvolmento anterior à implantação de *code review* possuia fases bem definidas e descritas, com informações e procedimentos disponíveis para os envolvidos. Para a implantação dos processos descritos neste trabalho, a documentação referente foi atualizada e novamente disponibilizada. A figura 4.1 ilustra o modelo proposto.

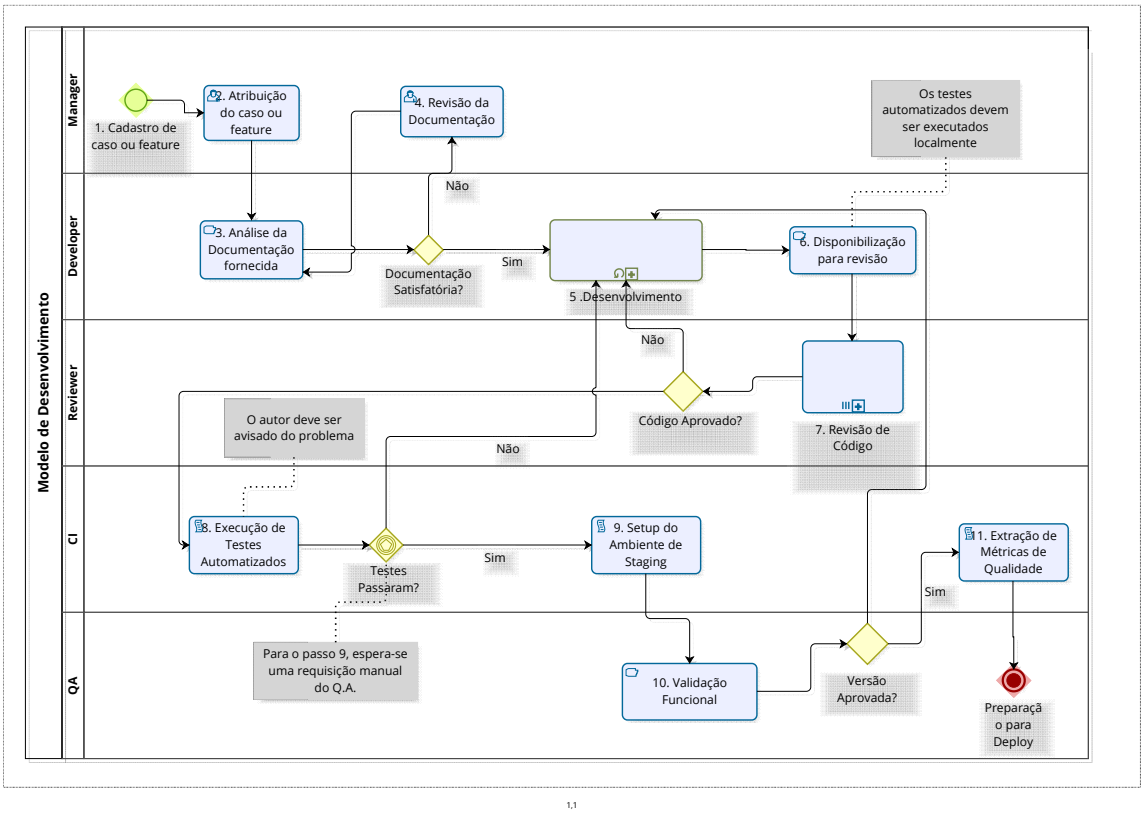

Figura 4.1: Diagrama do modelo de desenvolvimento droposto

O gerente, tem como principais papéis a atribuição das tarefas de desenvolvimento e a viabilização da documentação considerada adequada para o desenvolvedor. Muitas vezes executa com papel semelhante ao *Scrum Master*, atuando na negociação com os *stakeholders*. É o detentor da premissa do planejamento e da prerrogativa de decidir se uma tarefa deve ou não ser executada, e com qual prioridade. Esses papéis não estão diretamente relacionados ao objetivo deste experimento e não serão minunciados aqui.

O desenvolvedor, ao receber uma tarefa e sua respectiva documentação, deve

iniciar a execução da mesma. Nesta etapa, são escritos ou adaptados testes automatizados relacionados à tarefa. Não há, no modelo oficial de desenvolvimento, referência à utilização de ferramentas para análise estática do código em busca de aderência a padrões e defeitos de legibilidade e segurança, técnica conhecida como *linting*.

Sempre que o desenvolvedor obtiver uma porção do trabalho considerada estável, deve disponibilizá-la para o *code review*. Esses *commits*, relacionados à tarefa em andamento, devem ser agrupados semanticamente para facilitar a revisão e manter o grupo de revisores, evitando a entrada desnecessária de um novo revisor e uma reanálise das mudanças passadas.

A etapa de revisão de código é abordada na próxima seção com mais detalhes.

Quando a revisão é aprovada, o código pode ser submetido à *codebase* principal, onde serão executados os testes automatizados para inferir o funcionamento do código novo com o código desenvolvido pelos outros membros da equipe. Caso haja algum problema, geralmente relacionado a conflitos de código ou testes que falharam, o desenvolvedor é notificado.

O Q.A. (*Quality Assurance*) é profissional responsável por verificar e validar as novas versões do software. Buscando garantir aderência ao processo, documentação adequada e atenção do produto desenvolvido ao solicitado, o Q.A. pode, a qualquer momento, receber as modificações que passaram pelas etapas anteriores em seu ambiente de *staging* para validação funcional. Esse ambiente recebe tal denominação porque simula (ou encena) as configurações, riscos e características do ambiente de produção. Por exemplo, alterações na estrutura de dados não devem ocasionar inconsistências, enquanto mudanças de configuração de desenvolvimento para ambiente real não devem ocasionar falhas. Nesse ambiente, o Q.A. realiza os testes e verificações gerais na plataforma, observando principalmente se os requisitos foram satisfatoriamente cumpridos.

Se o Q.A. encontra algum problema, os desenvolvedores envolvidos com as mudanças são notificados. Caso contrário, uma nova versão é gerada e identificada por um código único. Esta versão deve ser analisada quanto à qualidade do código e outras métricas estruturais.

Por fim, a versão é empacotada para entrega e fica à disposição. Se haverá ou

não o *deploy*, é uma decisão do gerente, baseando-se nas modificações que fazem parte da versão e no planejamento do projeto.

4.1 Modelo para prática do *code review*

O processo de *code review* foi elaborado de acordo com diretrizes da organização e com práticas encontradas na literatura (ASUNDI; JAYANT, 2007; ACKERMAN; FOWLER; EBENAU, 1984; BAYSAL et al., 2012), dentre as quais podemos citar a organização de diversas unidades de código pertences a uma funcionalidade dentro da mesma revisão e o modelo de escolha e atribuição dos revisores. A figura 4.2 ilustra o modelo proposto. Recomendações específicas sobre a prática de *code review*, tanto para o autor quanto para o revisor, são apresentadas no capítulo 5.

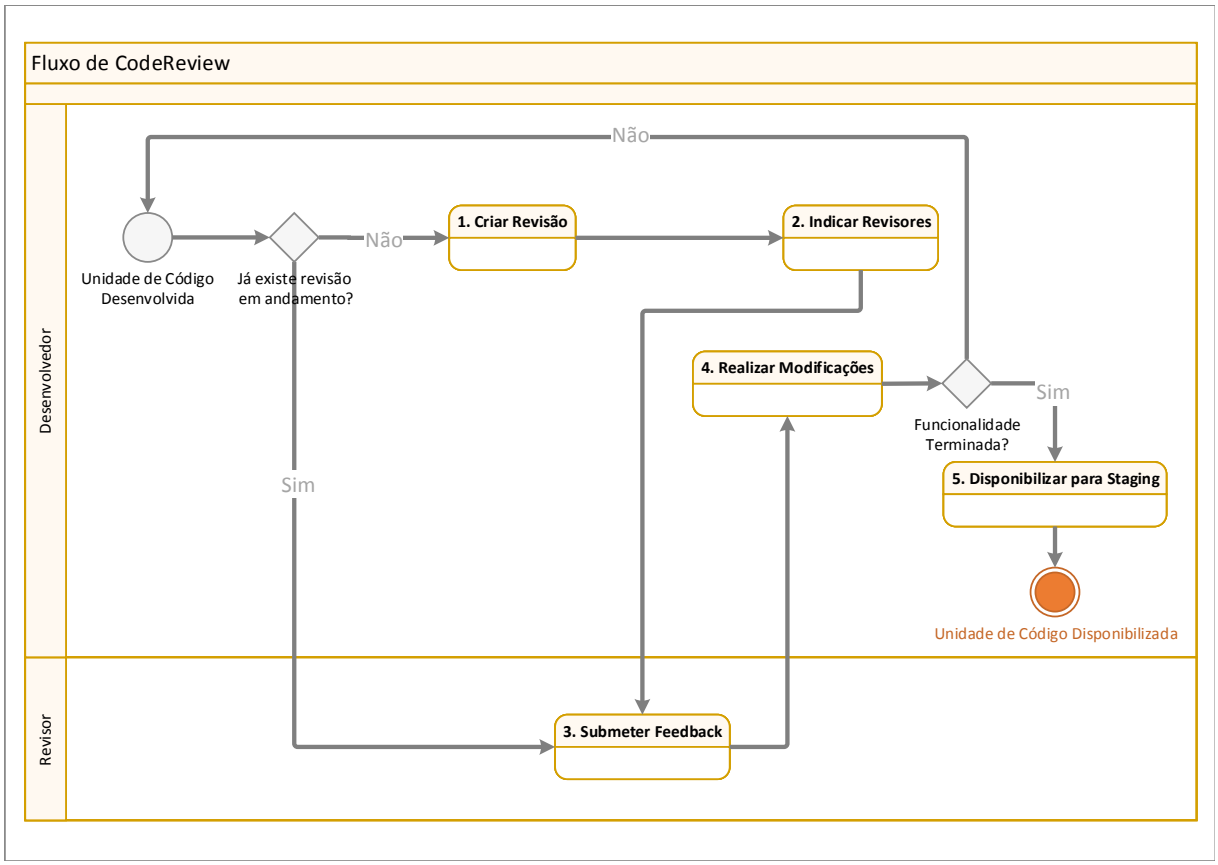

Figura 4.2: Diagrama do modelo de *code review* proposto

Sempre que possuir uma unidade de trabalho bem definida, o desenvolvedor deve disponibilizá-la para revisão. Esta unidade é representada por um *commit* no controle versão, enquanto o conjunto de *commits* relacionados à tarefa em execução constitui uma

branch. A revisão engloba toda a extensão da *branch*, entre o primeiro *commit* e a aprovação da mesma. Ou seja, se um novo *commit* for adicionado à *branch*, deve ser disponibilizado dentro da mesma revisão, e os revisores devem ser notificados desta mudança. Caso seja o primeiro *commit* da *branch*, o autor deve iniciar o *code review* e convidar os revisores.

Os revisores precisam ter acesso ao código e tecer comentários em partes específicas que compõem a mudança. Quando o revisor considerar que a mudança adere aos padrões de qualidade necessários, deve expressar seu consentimento de forma sistemática.

O autor deve refletir sobre os comentários do revisor, de acordo com as diretrizes e objetivos do *code review* aqui apresentados. Não há obrigatoriedade em aceitar e realizar as modificações indicadas, e existem diversos motivos para isso, como falta de tempo, custo/benefício inviável ou conhecimento de outras partes do sistema que o revisor não possui. Contudo, o autor deve se comprometer com a qualidade do código e utilizar as observações da melhor forma possível, registrando suas decisões e a memória do raciocínio para chegar em tais conclusões dentro da ferramenta de revisão.

Mudanças motivadas pela revisão devem ser submetidas como *commit* dentro da mesma revisão. Quando o autor sentir que existem revisões suficientes ou quando todos os revisores manifestaram sua aprovação, pode fechar a revisão e integrar sua modificação à *codebase*.

Em diversos modelos encontrados no mapeamento sistemático da literatura, é possível perceber que a revisão deve ser obrigatória e que não há possibilidade de integração do código à *codebase* principal sem o consentimento de um número mínimo de revisores. Apesar dos diversos argumentos que embasam tal modelo, decidiu-se por um *code review* não obrigatório, buscando flexibilidade no processo, diminuição de barreiras para implantação e atritos no lançamento de modificações urgentes.

4.2 Necessidade de novas ferramentas

Para a adoção deste novo modelo, percebeu-se a necessidade da adoção de novas ferramentas no processo de desenvolvimento:

- Uma ferramenta para apoiar a execução do *code review*, integrada ao controle de versão e ao ambiente de desenvolvimento;

- uma ferramenta para extração de métricas estruturais das novas versões de maneira contínua e com registro histórico para análise e comparação.

Nas próximas seções são apresentados tópicos relacionados às caracterísitcas, levantamento e definição das ferramentas.

4.3 Ferramenta para extração de métricas

Para a escolha da ferramenta para extração de métricas, foram elucidadas diversas características consideradas importantes, dado o contexto no qual o trabalho foi aplicado. Tais aspectos foram divididos em duas categorias: àqueles diretamente relacionados ao trabalho em desenvolvimento e ao modelo proposto; e outros ditados graças às diretrizes internas da organização, como a política de segurança e confidencialidade das informações. As características também foram classificadas com um indicador de prioridade, sendo que o menor número indica maior procedência sobre as outras. Um resumo dos requisitos é apresentado na tabela 4.1.

Tabela 4.1: Requisitos desejáveis para a ferramenta de extração de métricas

ID	Categoria	Prioridade	Descrição
RD.01	P	1	Permitir a execução automática das análises
RD.02	P	1	Apresentar métricas de manutenibilidade, legibilidade e defeitos de código
RD.03	P	1	Analisar código fonte em PHP e Java (Android)
RD.04	P	1	Permitir o registro e comparação do histórico de análises
RD.05	P	2	Apresentar métricas de complexidade e tamanho de código
RD.06	P	2	Permitir a customização das regras de análise
RD.07	O	2	Arquitetura *self-hosted*
RD.08	O	3	Permitir de execução remota da análise
RD.09	O	3	Projeto *OpenSource*
RD.10	O	4	Administração web

Para acompanhamento periódico do impacto do *code review* no processo de desenvolvimento, a ferramenta deve permitir a execução automática das análises, a partir de um evento como a construção de uma nova versão do software. Além disso, o registro e a comparação histórica são fundamentais para apoiar conclusões sobre a influência da prática. É também necessário que a ferramenta em questão apresente métricas relaciona-

das à qualidade do código fonte, como a manutenibilidade e legibilidade. Estes requisitos estão associados às hipóteses levantadas neste trabalho e a extração de tais métricas irá auxiliar a suportar ou refutá-las.

Para evitar a existência de falsos positivos e defeitos alheios aos padrões da organização, as regras utilizadas para classificar determinados padrões no código como defeituosos devem ser passíveis de personalização. De acordo com a organização, as ferramentas que possuem acesso ao código fonte devem ser instaladas dentro da infraestrutura de servidores de desenvolvimento e *build* da mesma. Esta arquitetura é composta por diversas máquinas virtuais, o que estimula a importância da comunicação remota entre o analisador de código fonte e o banco de dados ligado à interface de administração.

Na política da organização, é desejável que os softwares que fazem parte do fluxo de desenvolvimento sejam de código aberto, apesar desta não ser uma restrição. Da mesma forma, devido à topologia da infraestrutura disponível, é recomendável que a administração da ferramenta seja feita através de interface web.

4.3.1 Comparação das ferramentas de extração de métricas

Nossas pesquisas se deram através da internet, indicação de colegas da área e consultoria. Assim, foram encontradas as seguintes ferramentas para apoiar a implantação no processo: Synopsys[11], Teamscale[12], Kiuwan[13], SonarQube[14], Semmle[15], e SQUORE[16]. As páginas oficiais dos projetos foram escrutinadas, buscando a constatação da aderência destes aos requisitos desejáveis listados na tabela 4.1. O resultado deste processo é sumarizado na tabela 4.2.

Existem diversas ferramentas para inspeção e extração de indicadores de qualidade de código, as quais podem ser utilizadas em diferentes projetos e contextos. A maior parte das ferramentas encontradas permite a execução automática das análises, bem como a comparação histórica dos resultados. Esse modelo de trabalho é condizente com conceitos como inspeção contínua e acompanhamento sistemático da qualidade do

[11]https://www.synopsys.com/software-integrity.html
[12]https://www.cqse.eu/en/products/teamscale/landing/
[13]https://www.kiuwan.com/
[14]https://www.sonarqube.org/
[15]https://semmle.com/products/
[16]http://www.squoring.com/en/

Tabela 4.2: Características das ferramentas de extração de métricas

	Synopsys	**Teamscale**	**Kiuwan**	**SonarQube**	**Semmle**	**SQUORE**
RD.01		X	X	X	X	X
RD.02	X	X	X	X	X	X
RD.03	X	X	X	X	X	X
RD.04		X	X	X	X	X
RD.05	X	X	X	X	X	X
RD.06		X		X		X
RD.07		X		X		
RD.08		X	X	X	X	X
RD.09				X		
RD.10	X	X	X	X	X	X

código (PRAUSE; APELT, 2008). Todas as ferramentas analisadas também possuem administração web e apresentam métricas de qualidade e tamanho de software.

Utilitários como o SonarQube (único candidato *OpenSource*) e o TeamScale fornecem um leque maior de indicadores, como duplicação de código, dívida técnica e vulnerabilidades. Esses também são os únicos entre os candidatos a permitir a instalação dentro da infraestrutura da organização. Esses, junto ao SQUORE, permitem a customização das regras de análise.

4.3.2 Apresentação da ferramenta escolhida para extração de métricas

Dentre os candidatos apresentados, o SonarQube foi atendeu todos os requisitos desejáveis. O projeto está atualmente em sua versão 6.2 e pode ser utilizado como *self-hosted* ou online, tanto para projetos enterprise quanto *OpenSource*. Segundo o site oficial, o projeto possui atualmente cerca de 2 milhões de usuários em aproximadamente 50 mil organizações. O OpenHub[17] indica que o SonarQube utiliza principalmente a linguagem Java e conta atualmente com 121 contribuidores. São 4708 *commits* apenas nos últimos 12 meses, e 454 nos últimos 30 dias[18]. As visualizações dos indicadores proposta pelo projeto são baseadas no método SQALE[19]. Criado em 2010, propõe que sejam definidas regras para análise contínua do código fonte, evitando que o número de defeitos cresça de

[17]https://www.openhub.net
[18]https://www.openhub.net/p/sonar
[19]http://www.sqale.org/

forma descontrolada (LETOUZEY, 2012). O método possui quatro pilares: qualidade, análise, índices e indicadores.

No SonarQube, a qualidade é definida através da análise de requisitos não funcionais estruturais obtidos através do código. Tal extração é apoiada nas regras que indicam violações de código. Os índices representam os custos aferidos pelo débito técnico, enquanto os indicadores são expressões visuais destas informações relacionadas à outras, como histórico, número de linhas de código e complexidade.

A arquitetura do SonarQube pode ser apresentada em 4 componentes principais: 1) uma interface para administração e recuperação dos dados para os desenvolvedores e gerentes; 2) uma API para o suporte à navegação, fornecimento dos dados e integração com outros componentes e softwares; 3) o *scanner* que efetivamente executa a análise do código e os registra (via API) no banco de dados do SonarQube; 4) banco de dados no qual são armazenadas as configurações, o histórico de resultados e suas visualizações.

Os scanners podem ser executados no servidor de *build*/integração, realizando a análise dos projetos. Através de plugins é possível analisar dezenas de linguagens diferentes e realizar a integração com outros softwares. Outra caracterísitca importante do software é a integração com aplicações para administração de ciclo de vida de software. Sistemas como o Maven[20], Ant[21] e Gradle[22] e *engines* de integração contínua como Jenkins[23], MSBuild[24] e Bamboo[25] podem ser utilizados para disparar análises no SonarQube, integrando-o ao ciclo de desenvolvimento.

4.4 Ferramenta para execução do *code review*

A ferramenta de apoio ao processo de *code review* deve embasar a interação entre revisor e autor. Deve notificar os envolvidos quando novos eventos, como a adição de uma nova versão, comentários e resolução de pendências ocorrer. De maneira análoga ao apresentado na seção 4.3, os requisitos desejáveis foram priorizados e separados entre

[20] https://maven.apache.org/
[21] http://ant.apache.org/
[22] https://gradle.org/
[23] https://jenkins.io/
[24] https://msdn.microsoft.com/pt-br/library/dd393574.aspx
[25] https://br.atlassian.com/software/bamboo

Tabela 4.3: Requisitos desejáveis para a ferramenta de *code review*

Categoria	**Prioridade**	**Descrição**	
RD.01	P	1	Permitir a adição de novas modificações à mesma revisão
RD.02	P	1	Permitir que o revisor declare o *changeset* como aprovado
RD.03	P	1	Integração com o Git
RD.04	P	2	Possibilitar a escrita de comentários em partes específicas do código
RD.05	P	2	Integração com IDEs
RD.06	P	2	API para consolidação dos resultados
RD.07	P	2	Gráficos e visualizações sobre cobertura de revisão
RD.08	O	2	*self-hosted*
RD.09	P	3	Notificações sobre novas modificações, comentários e resoluções
RD.10	O	3	*OpenSource*
RD.11	O	4	Administração web

diretrizes da organização e em caracterísitcas relevantes para a execução do trabalho. As ferramentas candidatas foram avaliadas quanto à capacidade de atender o modelo de *code review* proposto na seção 4.1, o qual foi baseado na literatura relacionada a este trabalho. Os requisitos desejáveis são apresentados na tabela 4.3.

A integração da ferramenta de *code review* com o sistema de controle de versão da organização é fundamental para automação do processo de notificação e revisão. A adição de novas mudanças na mesma revisão auxilia no processo de submissão de novas alterações e propicia que revisores próximos da análise em andamento continuem na tarefa.

A decisão de aceitar uma mudança na *codebase* deve ser declarada pelo revisor, e o utilitário escolhido deve permitir isso. Para contribuir contextualização das indicações do revisor e assim amparar o decurso de correção do desenvolvedor, estas observações devem ser associadas diretamente a parte do código em revisão. A integração com as IDEs de desenvolvimento diminuem barreiras de implantação e podem aumentar a eficiência de revisão e diminuir o tempo de resposta.

A possibilidade de APIs públicas é importante para auxiliar no processo de extração das informações referentes à prática do *code review*. Análogo ao disposto na seção 4.3, algumas políticas da organização incentivam a adoção de softwares *OpenSource* e de administração web, além de exigir que aplicações com acesso ao código fonte sejam instaladas dentro da infraestrutura interna disponibilizada para tais fins.

4.4.1 Comparação das ferramentas de *code review*

Nossas pesquisas se deram através da internet, indicação de colegas da área e consultoria. Os candidatos escolhidos foram: Codacy[26], Crucible[27], Gerrit[28], Upsource[29], Phabricator[30] e ReviewBoard[31]. Com as informações encontradas nas documentações oficiais dos projetos, a tabela 4.4 representa a aderência das ferramentas aos requisitos desejáveis sumarizados na tabela 4.3.

Todas as ferramentas canditadas permitem a integração com o Git, comentários em trechos específicos do código, adição de mudanças em revisões existentes e notificações para os envolvidos no processo. Algumas, Como o Codacy e o Crucible permitem notificações para plataformas de comunicação de mercado, como o Slack[32] e o HipChat[33]. Já o Upsource permite notificações através de integração com IDEs. O fornecimento de APIs para integração e extração de informações também é funcionalidade comum a todas as ferramentas analisadas.

Tabela 4.4: Características das ferramentas de *code review*

	Codacy	**Crucible**	**Gerrit**	**Upsource**	**Phabricator**	**ReviewBoard**
RD.01	X	X	X	X		X
RD.02	X	X	X	X	X	X
RD.03	X	X	X	X	X	X
RD.04	X	X	X	X	X	X
RD.05			X	X		X
RD.06	X	X	X	X	X	X
RD.07	X	X		X	X	
RD.08			X	X	X	X
RD.09	X	X	X	X	X	X
RD.10			X		X	X
RD.11	X	X	X	X	X	X

[26]https://www.codacy.com/
[27]https://www.atlassian.com/software/crucible
[28]https:// www.gerritcodereview.com/
[29]https://www.jetbrains.com/upsource/
[30]https://www.phacility.com/
[31]https://www.reviewboard.org/
[32]https://slack.com/
[33]https://www.hipchat.com/

4.4.2 Apresentação da ferramenta escolhida para extração de métricas

Os projetos Gerrit, ReviewBoard e Upsource fornecem a mesma quantidade de requisitos desejáveis listados na tabela 4.3. Enquanto os dois primeiros são *OpenSource*, eles não apresentam as visualizações de dados encontradas no último. Assim, considerando as prioridades definidas para cada um das caracterísitcas utilizadas na comparação, o Upsource foi o escolhido para apoiar o *code review* no processo de desenvolvimento estudado.

O Upsource, mantido pela JetBrains[34], oferece funcionalidades que suportam diversos modelos de *code review*. Notoriamente, a desenvolvedora do produto aponta duas formas de utilização: um modelo ágil, onde a equipe é responsável por decidir quais modificações precisam de revisão, ou a utilização de filtros e gatilhos automáticos disparadas por alterações em determinadas partes do projeto para determinar e notificar os revisores. é possível mesclar as duas abordagens e criar um modelo adaptado ao contexto no qual o sistema é inserido. Além disso, o software é capaz de indicar revisores para determinada alteração, com base no histórico de ação dos envolvidos.

São fornecidos plugins para diversas IDEs, para a realização do *code review* e recebimento de notificações. Da mesma forma, é possível adicionar novas revisões ao *code review* já efetuado, manualmente ou mencionando o identificador único do *Code Review* na mensagem do *commit*. Outra funcionalidade chave do software que apoia o modelo proposta é a revisão de *branches* inteiras ou os *pull-requests* solicitados. Desta forma, novas revisões são adicionadas à revisão automaticamente.

O suporte à visualização e recuperação da informação é peça fundamental da ferramenta. Existem gráficos que mostram a relação entre os desenvolvedores e revisores, auxiliando no proceso de alocação desses no futuro. Outra representação notável é a relação entre modificações na *codebase* e o número de revisões, indicando a cobertura de código que é revisado, por módulo ou pelo projeto completo. Também é possível comparar o número de reviews concluídos e abertos, e a adesão dos desenvolvedores à prática.

Desde a versão 1.0, a aplicação apresenta suporte à diversas linguagens e tipos de repositórios, comparação entre diferentes versões e *branches* do código. A partir da

[34]https://www.jetbrains.com/

versão 2.0, a integração com as principais IDEs (através de plugins) foi disponibilizada, em conjunto ao mecanismo para indicação de revisores. Na versão 3.5 o a funcionaliade de busca do sistema foi melhorado, alcançando arquivos entre projetos e pessoas e tornando o fluxo de trabalho mais flexível e abrangente. Nesta versão novas visualizações e gráficos foram acrescentados. O Upsource é construído primariamente em Java, e conta com o SGBD Apache Cassandra[35].

[35]http://cassandra.apache.org/

5 Metodologia e Implantação do *Code Review*

A condução de experimentos científicos em ambiente industrial enfrenta obstáculos, como a escassez de recursos como tempo e esforço para a avaliação dos métodos e resultados além da dificuldade de se conduzir experimentos controlados, princpalmente levando em consideração o custo (SJOBERG et al., 2002). Com tais desafios em mente, projetou-se o processo de implantação do *code review* e a coleta dos resultados. Este capítulo relata tais experiências.

5.1 Introdução do *code review* no cenário existente

Antes da implantação do *code review*, conduziu-se um levantamento para averiguar o nível de aceitação da equipe em relação à prática. Esse passo nos ajudou a entender as necessidades dos envolvidos no projeto e criar empatia em relação ao projeto, uma vez que a aderência ao modelo proposto seria somente possível com a adesão dos envolvidos.

A elaboração do modelo proposto no capítulo 4 foi realizada a participação dos pesquisadores, gerentes e desenvolvedores. Uma semana antes da entrega das últimas versões dos componentes sem *code review*, foi realizado um treinamento para a explicação do modelo e da implantação. Cerca de 15 dias antes do treinamento, a documentação do processo de desenvolvimento atualizado foi disponibilizada para os envolvidos, que puderam tirar dúvidas e tecer sugestões durante esse período. Essas observações foram, discutidas e levadas em consideração para adequações no processo proposto.

No treinamento, foram abordados objetos como fluxo de trabalho, objetivos, o que procurar durante a prática e como conduzir a interação, do ponto de vista do autor e do revisor. Estas diretrizes foram criadas com o objetivo mitigar possíveis problemas sociais atribuídos à prática e controlar melhor as variáveis do experimento.

As principais diretrizes para a condução do *code review* por parte do revisor foram:

- Realizar *reviews* curtas, e foque nos problemas prioritários. é normal que revisões terminem sem nenhuma observação;
- criticar ideias, não pessoas;
- não revisar grandes porções código de uma vez, trabalhe em pequenos lotes de até 200 linhas e evite o acúmulo;
- ter em mãos uma lista de problemas comuns;
- evitar passar mais de uma hora revisando código;
- adotar uma postura positiva e seja empático;
- estar pronto para mudar de ideia.

Já na perspectiva do autor, foram tratadas as seguintes orientações:

- Receber as críticas de forma construtiva;
- disponibilizar o código fonte em pequenos lotes e com prazos condizentes com o projeto;
- ter certeza que o código está devidamente organizado e comentado antes da disponibilização;
- manter a cabeça aberta e pronto para melhorar.

Uma pequeno guia foi desenvolvido, em conjunto com os desenvolvedores, para servir como base para a operação do *code review*. Este resumo em um conjunto de indicações que o desenvolvedor deve se atentar durante a prática:

- Os *linebreaks*, tamanhos de linha, posição das chaves estão de acordo com o padrão do projeto?
- As declarações de variáveis estão visíveis e semanticamente agrupadas?
- Os nomes de métodos, variáveis e classes aderem ao padrão do projeto?
- Há testes automatizados escritos para este código?

- Os testes escritos cobrem satisfatoriamente o código?
- Trechos mais complexos foram testados com mais atenção?
- O código apresenta uma boa solução para o problema proposto?
- Princípios *SOLID* e padrões de projeto foram aplicados?
- O código se encontra no componente (método, módulo, classe) correto?
- Há reaproveitamento de código existente, se possível?
- Há indícios de *over-engineering*?
- Os nomes de métodos, variáveis e classes fazem sentido a primeira vista?
- É possível imaginar o funcionamento do código apenas pela leitura?
- As mensagens de *log*, comentários e exceções são compreensíveis?
- Existem inconsistências claras com a regra de negócio do projeto?
- Existem condicionais invertidas, não previsão de entradas particulares ou exceções mal tratadas que podem levar a um comportamento inesperado?
- Há erros ortográficos ou de compreensibilidade nas mensagens para os usuários?
- Existem estruturas/construções da linguagem conhecidas como vulnerabilidades?
- As entradas de usuário estão sendo sanitizadas e verificadas contra ataques?
- Existe algum indício de *deadlock*?
- Existem *leaks* de memória ou outros recursos?
- É possível melhorar o desempenho de consultas (*networking* ou *SQL*)?
- Existem chamadas custosas dentro de *loops* de tamanho indefinido/volátil?

5.2 Coleta de resultados

Para avaliar o impacto do *code review* nos fatores dispostos na seção 3.2, foram aplicadas duas técnicas: a extração de métricas quantitativas de qualidade estrutural das versões antes de depois da implantação da prática e a condução de uma *survey* para avaliar a ótica dos desenvolvedores sobre os resultados do *code review*.

O primeiro passo para a extração de métricas foi a construção de um *script* para recuperação das métricas em versões antigas. Para cada versão dos componentes o *script* configurou e executou o *scanner* do SonarQube, traçando o histórico das características relevantes. Os dados da tabela 3.1, assim como outros que serão apresentados no captítulo 6, foram auferidos desta forma. A avaliação do impacto do *code review* na qualidade de código foi feita através dos indicadores de débito técnico, complexidade ciclomática, número de defeitos (*issues*) e duplicação de código, que são métricas disponibilizadas pela ferramenta escolhida SonarQube. Para este trabalho foram selecionados os seis meses anteriores ao *code review* e os seis meses de observação após a implantação do mesmo. Apenas duas versões do Projeto APP1 foram lançadas depois do início do experimento, o que nos levou a descartar suas amostras.

A *survey* foi aplicada com objetivo de mensurar impactos menos objetivos do trabalho, mas que a literatura relacionada indica como possivelmente relacionados ao *code review* (BACCHELLI; BIRD, 2013). Assim, as questões abordadas buscam investigar o impacto da prática na difusão do conhecimento, senso de responsabilidade compartilhado, discussão e implantação de boas soluções. Tal método também foi aplicado para avaliar as mitigações dos aspectos negativos relacionados ao *code review*.

6 Resultados Obtidos

Os resultados obtidos foram divididos em duas categorias, para facilitar a explicação: A primeira, voltada para as métricas de qualidade não funcional e observação do impacto das técnicas aplicadas na qualidade do código. A segunda é composta pelos resultados do questionário aplicado aos desenvolvedores que participaram do desenvolvimento, com o intuito de examinar o impacto do *code review* em aspectos que não podem ser diretamente inferidos pela análise do código.

6.1 Métricas de qualidade não funcional

Durante o experimento, em cada lançamento de uma nova versão foram extraídas as métricas de interesse do SonarQube através da Web API[36] disponibilizada. Foram elas:

1. Linhas de Código (LOC)
2. Porcentagem de Linhas Duplicadas (D)
3. Débito Técnico (DT)
4. Defeitos (Issues) (I)
5. Complexidade Ciclomática (CC)

Para fins de comparação com o estado prévio dos projetos, utilizou-se dados de 6 meses anteriores à implantação do *code review*. Observou-se então essas métricas (M) em dois aspectos distintos: média no projeto (6.1) e taxa de variação (6.2). A primeira como *proxy* para avaliar se a qualidade geral do software foi afetada de acordo com a prática de *code review*, e a segunda para observar se a *velocidade* de evolução de determinada faceta do sistema foi alterada de acordo com a variação da quantidade de linhas de código (LOC). O estudo da variação pode indicar efeitos da prática na contenção do crescimento das métricas que, devido ao curto período de avaliação, não seriam visíveis através da

[36] https://docs.sonarqube.org/display/DEV/Web+API

análise por média. A exceção se dá por conta da porcentagem de linhas duplicadas (D), que já está relacionada com o total de LOC e não passa pelo tratamento definido em 6.1.

Nas próximas seções as métricas são apresentadas pelas siglas acima, em conjunto com um "V" ou "M" para identificar se foi abordado o aspecto de variação ou médio. Por exemplo, a variação de Débito Técnico será taxada como DT.V, e a média de defeitos como I.M.

$$Med(\mathrm{M}) = \frac{\mathrm{M}}{\mathrm{LOC}} \tag{6.1}$$

$$Var(\mathrm{M}_{antes}, \mathrm{M}_{depois}) = \frac{\mathrm{M}_{depois} - \mathrm{M}_{antes}}{\mathrm{LOC}_{depois} - \mathrm{LOC}_{antes}} \tag{6.2}$$

6.1.1 Metodologia de verificação da significância estatísitca

Para estabelecer juízo de valor sobre o experimento é necessário, primariamente, avaliar a relevância estatísitica sobre os dados coletados durante sua execução. A literatura acerca deste assunto indica possível subutilização de ferramentas estatísitcas adequadas na Engenharia de Software (MILLER et al., 1997), o que pode acarretar em trabalhos com resultados questionáveis e sem fundamentação sólida (DYBÅ; KAMPENES; SJØBERG, 2006).

As análises dispostas nesta seção foram conduzidas de acordo com o trabalho de Araújo et al. (2006), com objetivo de observar se a implantação da prática de *code review* no processo de desenvolvimento da organização alvo surtiu efeitos estatísticamente significativos nas métricas estruturais dos projetos alvo. O nível de significância (valor p ou *p-value*) escolhido foi 0,05, muito utilizado na Engenharia de Software e outras ciências naturais.

Levando em consideração o grande número de amostras analisadas nestra trabalho, não são demonstrados os passos de cada uma das análises, para manter o tamanho do trabalho dentro de um limite razoável. Nesta seção são utilizadas amostras de terceiros para ilustrar os tratamentos estatísticos utilizados, enquanto na seção 6.1.2 os resultados são apresentados em forma de tabela.

O primeiro passo é verificar a normalidade dos dados. Como pode-se observar

que todas métricas contém menos de 30 amostras, aplica-se o teste de *Shapiro-Wilk*. São consideradas as seguintes hipóteses:

- H0 (hipótese nula): as amostras apresentam distribuição normal;
- H1 (hipótese alternativa): as amostras não apresentam distribuição normal.

Caso o *p-value* do teste seja maior que 0,05, deve-se aceitar a hipótese nula, uma vez que não há indícios que apontem para a hipótese alternativa. A figura 6.1 mostra o resultado deste tipo de teste.

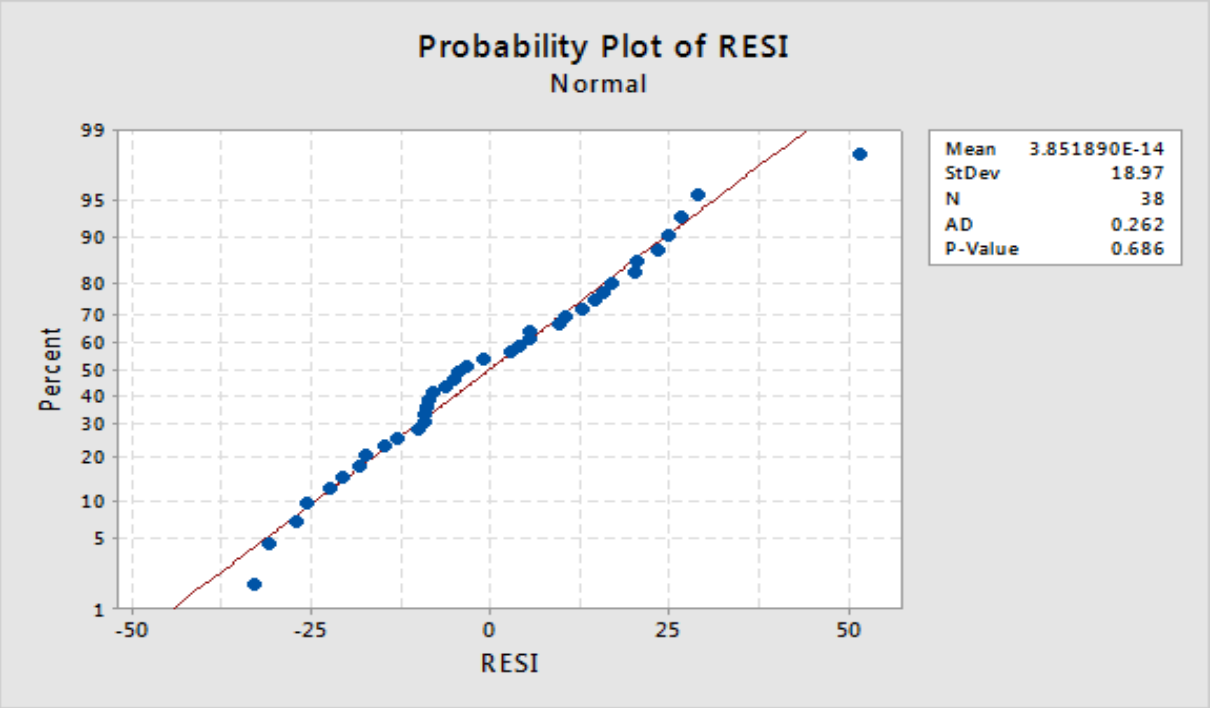

Figura 6.1: Gráfico representando amostra normal segundo o teste de *Shapiro-Wilk* (Pennsylvania State University, 2017b)

Da mesma forma, verifica-se se as amostras são homocedásticas. Através do teste de *Levene*, observou-se o *p-value* e foram avalaidas duas hipóteses:

- H0 (hipótese nula): as amostras homocedásticas.
- H1 (hipótese alternativa): as amostras não são homocedásticas;

Ao verificar resultado maior que a significância estabelecida de 5%, aceita-se a hipótese de que os dados são homocedásticos. É o caso da amostra representada pela figura 6.2

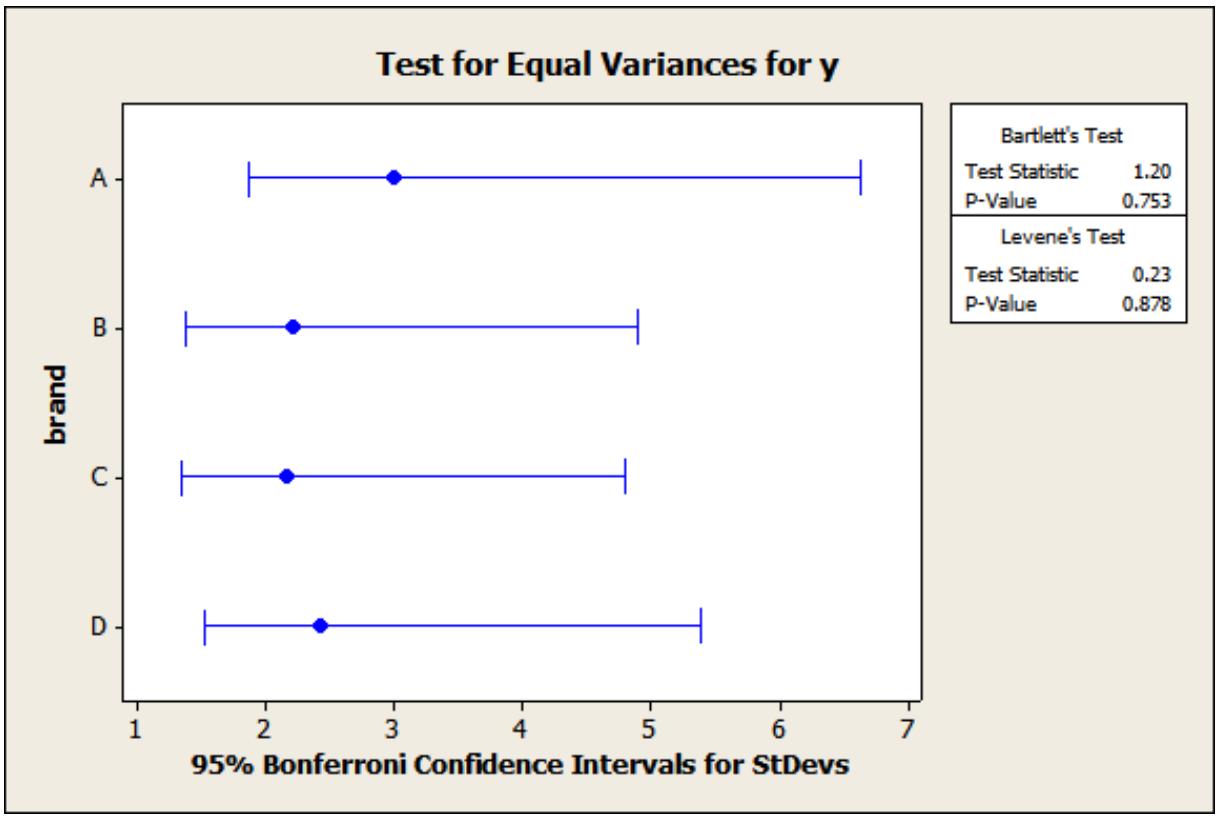

Figura 6.2: Teste de *Levene* apontando amostra homocedástica (WANG, 2009)

Ao garantir que as amostras são normais e homocedásticas, pode-se aplicar um teste paramétrico para verificar se as médias das amostras antes e depois do *code review* são significativamente distintas, do ponto de vista estatístico. Elegeu-se o *Teste T* com os fatores "antes" (ACR) e "depois" (DCR) da aplicação do *code review*. São as hipóteses:

- H0 (hipótese nula): não há diferença entre as médias;
- H1 (hipótese alternativa): há diferença entre as médias.

Um *p-value* maior que 0,05 indica que deve-se descartar a hipótese nula, como é exemplificado na figura 6.3. Caso os dados não sejam normais e homocedásticos, aplica-se testes não paramétricos para chegar a esta conclusão. Para estas situações aplica-se o teste de *Mann-Whitney*, o que pode ser visto na figura 6.4.

Paired T-Test and CI: Before, After

```
Paired T for Before - After

              N      Mean     StDev   SE Mean
Before       10   5.38000   3.20583   1.01377

After        10   4.86000   3.10312   0.98129
Difference   10  0.520000  0.407704  0.128927


95% lower bound for mean difference: 0.283662
T-Test of mean difference = 0 (vs > 0): T-Value = 4.03  P-Value = 0.001
```

Figura 6.3: *Teste T* para asserção da significância estatística da amostra (Pennsylvania State University, 2017a)

Mann-Whitney Test and CI: Female_Multitask, Male_Multitask

```
                   N  Median
Female_Multitask  10   75.00
Male_Multitask    10   55.00


Point estimate for ETA1-ETA2 is 10.00
95.5 Percent CI for ETA1-ETA2 is (-9.99,30.01)
W = 120.0
Test of ETA1 = ETA2 vs ETA1 > ETA2 is significant at 0.1365
The test is significant at 0.1271 (adjusted for ties)
```

Figura 6.4: Teste de *Mann-Whitney* para asserção da significância estatística da amostra (FROST, 2013)

6.1.2 Apresentação dos resultados

Os resultados obtidos em cada projeto e os resultados dos métodos estatísticos aplicados são apresentados nesta seção em tabelas, separados por projeto. As amostras com relevância estatística são destacadas em negrito, e apresenta-se suas médias aritiméticas dos indicadores nos dos dois períodos estudados (antes e depois do *code review*) e a comparação entre tais valores.

O *Backend*, retratado na tabela 6.1, é o maior projeto que participou do experimento, em número de linhas de código. É escrito primariamente em PHP e apresenta

a maior taxa de crecimento nesse aspecto, mais que dobrando seu tamanho no período observado. A aplicação do teste de *Shapiro-Wilk* mostrou que, dentre as métricas obtidas, apenas a dívida técnica média (DT.M) e a variação de linhas duplicadas (D.V) não foram tidas como normais, e por isso foram submetidas ao teste não paramétrico de *Mann-Whitney* Para as outras métricas executou-se o teste de *Levene*, confirmando integralmente a hipótese nula de que são homocedásticos.

Tabela 6.1: Resultados do projeto *backend*

Versão	Antes/depois	LOC	CC.M	CC.V	**D**	D.V	DT.M	**DT.V**	I.M	**I.V**
3,5	ACR	14494	0,12529	*	**3,38%**	*	0,12288	*	0,01635	*
3,6	ACR	16619	0,12612	0,13176	**4,15%**	3,63%	0,15109	**0,34353**	0,01817	**0,03059**
3,7	ACR	16836	0,12675	0,17512	**3,86%**	-13,41%	0,15491	**0,44700**	0,01823	**0,02304**
3,8	ACR	17825	0,13010	0,18706	**3,42%**	-4,43%	0,15663	**0,18605**	0,01835	**0,02022**
3,9	ACR	20338	0,13487	0,16872	**3,15%**	-1,10%	0,16786	**0,24751**	0,02080	**0,03820**
4,0	ACR	23656	0,13307	0,12206	**2,87%**	-0,82%	0,17010	**0,18385**	0,02021	**0,01658**
4,1	DCR	24581	0,13336	0,14054	**2,56%**	-3,37%	0,16285	**-0,02270**	0,01993	**0,01297**
4,2	DCR	26674	0,13530	0,15815	**2,51%**	-0,24%	0,16724	**0,21882**	0,01938	**0,01290**
4,3	DCR	28231	0,12961	0,03211	**2,37%**	-0,89%	0,16400	**0,10854**	0,01870	**0,00706**
4,4	DCR	29447	0,12728	0,07319	**2,24%**	-1,09%	0,15757	**0,00822**	0,01796	**0,00082**
4,5	DCR	31025	0,12364	0,05577	**2,29%**	0,30%	0,15394	**0,08619**	0,01779	**0,01458**
Shapiro-Wilk			>0,1	>0,1	**>0,1**	<0,010	<0,010	**>0,1**	>0,1	**>0,1**
Levene			0,863	0,286	**0,124**	*	*	**0,713**	0,497	**0,494**
Teste T			0,864	0,057	**0,002**	*	*	**0,018**	0,929	**0,013**
Mann-Whitney			*	*	*	0,5309	0,2963	*	*	*
Média ACR			*	*	**3,47%**	*	*	**0,28159**	*	**0,02573**
Média DCR			*	*	**2,40%**	*	*	**0,07981**	*	**0,00967**
Proporção Antes/Depois			*	*	**69%**	*	*	**28,34%**	*	**37,58%**

O *Teste T* apontou que as médias das linhas duplicadas (D), dívida técnica variável (DT.V) e variação de defeitos (I.V) foram aceitos na hipótese alternativa de que são significativamente diferentes. As proporção entre as médias dos períodos observados aponta que o software sofreu um decréscimo de 31% na duplicação de código, enquanto a variação da dívida técnica caiu cerca de 71% e a variação de defeitos, 62%.

No projeto API1, apenas as amostras de variação da duplicação e dos defeitos não foram identificados como normais. Para essas foi utilizado um teste não paramétrico, mas que não apontou indícios para rejeição da hipótese nula de que as médias são equivalentes. Já a complexidade ciclomática média (CC.M), duplicação (D) e a média da dívida técnica (DT.M), além de normais e homocedásticas, apresentaram indícios de significância através do *Teste T*. Este projeto é escrito primariamente e PHP e serve como API para prover serviços à softwares de terceiro e uma das plataformas *mobile* do produto.

Percebe-se uma diminuição ligeira do CC.M em 2,28% e um decréscimo de 40% da duplicação de linhas código. Já a variação da dívida técnica caiu em cerca de 10%. Esses resultados podem ser vistos na tabela 6.2.

Tabela 6.2: Resultados do projeto API1

Versão	Antes/depois	LOC	**CC.M**	CC.V	**D**	D.V	**DT.M**	DT.V	I.M	I.V
3,5	ACR	6661	**0,15808**	*	**8,8%**	*	**0,16829**	*	0,01576	*
3,7	ACR	6785	**0,15814**	0,16129	**8,6%**	-0,16129	**0,17288**	0,41935	0,01710	0,08871
3,8	ACR	7240	**0,15373**	0,08791	**11,5%**	0,63736	**0,15608**	-0,09451	0,01671	0,01099
3,9	ACR	7334	**0,15435**	0,20213	**11,4%**	-0,10638	**0,15476**	0,05319	0,01663	0,01064
4,0	ACR	7396	**0,15441**	0,16129	**10,7%**	-1,12903	**0,14927**	-0,50000	0,01663	0,01613
4,1	DCR	7197	**0,15479**	0,14070	**7,4%**	1,65829	**0,14603**	0,26633	0,01653	0,02010
4,2	DCR	7218	**0,15434**	0,00000	**5,8%**	-7,61905	**0,14838**	0,95238	0,01690	0,14286
4,3	DCR	7414	**0,15080**	0,02041	**5,8%**	0,00000	**0,14567**	0,04592	0,01646	0,00000
4,4	DCR	7517	**0,15192**	0,23301	**5,7%**	-0,09709	**0,14367**	0,00000	0,01623	0,00000
4,5	DCR	7818	**0,14914**	0,07973	**5,6%**	-0,03322	**0,13981**	0,04319	0,01624	0,01661
Shapiro-Wilk			**>0,1**	>0,1	**>0,1**	<0,01	**>0,1**	>0,1	>0,1	<0,01
Levene			**0,844**	0,229	**0,187**	*	**0,157**	0,889	0,64	*
Teste T			**0,044**	0,275	**0,001**	*	**0,029**	0,306	0,72	*
Mann-Whitney			*****	*	*****	0,543		*	*	1
Média ACR			**0,15574**	*	**10,2**	*	**0,16026**	*	*	*
Média DCR			**0,15220**	*	**6,06**	*	**0,14471**	*	*	*
Proporção Antes/Depois			**97,72%**	*	**59,41%**		**90,30%**	*	*	*

Os resultados do projeto API2 podem ser vistos na tabela 6.3. Este foi o projeto com menor crescimento em linhas de código que participou do experimento. O teste de *Shapiro-Wilk* mostrou que apenas CC.M, CC.V E D são amostras normais. Contudo, o teste de *Levene* mostrou que a variação da complexidade ciclomática não é homocedástica, o que a classifica, junto com as amostras que não foram consideradas normais, para um teste não paramétrico. Assim como a API1, esse projeto é desenvolvimento primariamente em PHP e provê serviços para integração de softwares de terceiros e às plataformas *mobile* do produto.

O teste de *Mann-Whitney* indicou que, para as amostras de complexidade ciclomática média, deve-se aceitar a hipótese alternativa de que as médias não são iguais. Ou seja, há diferença, to ponto de vista estatístico, nos resultados antes e depois do *code review*. De maneira análoga, percebeu-se tal relevância através do *Teste T* para as amostras de duplicação de código. No primeiro, observou-se uma leve redução de 3,32%, e no segundo de 5,24%.

O APP1 (tabela 6.4) é desenvolvido em Java para a plataforma *mobile* Android.

Tabela 6.3: Resultados do projeto API2

Versão	Antes/depois	LOC	**CC.M**	CC.V	**D**	D.V	DT.M	DT.V	I.M	I.V
3,5	ACR	6390	**0,16839**	*	**31,1**	*	0,21049	*	0,01440	*
3,6	ACR	6391	**0,16836**	0	**31,1**	0	0,21045	0	0,01440	0
3,7	ACR	6486	**0,16836**	0,16842	**30,7**	-0,42105	0,20953	0,14737	0,01480	0,04211
3,8	ACR	6508	**0,16825**	0,13636	**30,7**	0	0,20805	-0,22727	0,01460	-0,04545
3,9	ACR	6509	**0,16823**	0	**30,7**	0	0,20802	0	0,01460	0
4,0	ACR	6858	**0,16740**	0,15186	**29,8**	-0,25788	0,21639	0,37249	0,01721	0,06590
4,1	DCR	7009	**0,16151**	-0,10596	**29,2**	-0,39735	0,21002	-0,07947	0,01712	0,01325
4,2	DCR	7027	**0,16237**	0,5	**29,1**	-0,55556	0,20948	0	0,01708	0
4,3	DCR	7064	**0,16322**	0,32432	**29**	-0,27027	0,20909	0,13514	0,01713	0,02703
4,4	DCR	7065	**0,16320**	0	**29**	0	0,20481	-30	0,01713	0
Shapiro-Wilk			**0,015**	>0,1	**>0,1**	>0,1	0,039	<0,010	<0,010	>0,100
Levene			*	0,013	**0,292**	0,729	*	*	*	0,218
Teste T			*		**0**	0,297	*	*	*	0,91
Mann-Whitney			**0,0142**	0,8852	*	*	0,3374	0,3832	0,1098	*
Média ACR			**0,16817**	*	**30,68333**	*	*	*	*	*
Média DCR			**0,16258**	*	**29,075**	*	*	*	*	*
Proporção Antes/Depois			**96,68%**	*	**94,76%**	*	*	*	*	*

Também apresentou crescimento expressivo durante o experimento, com um acréscimo de cerca de 75% em linhas de código. Todas as amostras foram classificadas normais e homocedásticas, com exceção das duplicações de código (D).

Tabela 6.4: Resultados do projeto APP1

Versão	Antes/depois	LOC	**CC.M**	CC.V	**D**	D.V	DT.M	DT.V	I.M	I.V
1,4	ACR	8307	**0,15734**	*	**2**	*	0,18286	*	0,02889	*
1,5	ACR	8560	**0,15806**	0,18182	**2,3**	0,11858	0,18586	0,28458	0,03014	0,07115
1,6	ACR	8612	**0,15850**	0,23077	**2,3**	0	0,18532	0,09615	0,03019	0,03846
1,7	ACR	9611	**0,15857**	0,15916	**3,7**	0,14014	0,19634	0,29129	0,03101	0,03804
1,8	ACR	9908	**0,16017**	0,21212	**3,6**	-0,03367	0,19298	0,08418	0,03129	0,04040
1,9	ACR	12007	**0,15216**	0,11434	**3,8**	0,00953	0,18556	0,15055	0,03057	0,02716
1,10	DCR	12879	**0,15048**	0,12729	**4,2**	0,04587	0,19256	0,28899	0,03176	0,04817
1,10,1	DCR	13083	**0,15165**	0,22549	**4,2**	0	0,18520	-0,27941	0,03012	-0,07353
1,11	DCR	13475	**0,15265**	0,18622	**4,1**	-0,02551	0,19050	0,36735	0,03124	0,06888
1,12	DCR	13844	**0,15299**	0,16531	**4**	-0,02710	0,18470	-0,02710	0,03012	-0,01084
1,13	DCR	14563	**0,15402**	0,17385	**3,8**	-0,02782	0,17799	0,04868	0,02857	-0,00139
Shapiro-Wilk			**>0,100**	>0,100	**0,034**	0,33	>0,100	>0,100	>0,100	0,088
Levene			**0,575**	0,537	*	0,232	0,843	0,149	0,538	0,094
Teste T			**0,005**	0,881	*	0,209	0,572	0,45	0,984	0,23
Mann-Whitney			*	*	**0,0101**	*	*	*	*	*
Média ACR			**0,15747**	*	**2,95%**	*	*	*	*	*
Média DCR			**0,15236**	*	**4,06%**	*	*	*	*	*
Proporção Antes/Depois			**96,76%**	*	**137,63%**	*	*	*	*	*

No caso dessa amostra, foi executado o teste não paramétrico de *Mann-Whitney* e constatou-se a relevância estatística do seu crecimento em 37,63%. No caso da complexidade ciclomática média, (CC.M), o *Test T* apontou um leve decréscimo de 3,24%. Nas

outras amostras não foi possível detectar variação estatisticamente significativa.

Estas métricas podem ser utilizadas para avaliação dos resultados em relação aos objetivos propostos na seção 3.2. No panorama da qualidade do código, os resultados obtidos convergeram com a literatura relacionada (KEMERER; PAULK, 2009; MCINTOSH et al., 2014; MORALES; MCINTOSH; KHOMH, 2015; BAVOTA; RUSSO, 2015). Em três dos quatro projetos analisados foi possível observar impactos positivos em duplicação de código e na complexidade ciclomática média. Enquanto na primeira métrica os resultados variaram em uma redução de 3,32% a 31%, na segunda observou-se decréscimo entre 2,28% e 5,24%. O projeto *Backend* foi o que mais apresentou melhoria nas métricas mensuradas. Esta constatação pode estar relacionada ao fato desse ser o projeto mais ativo do experimento, tendo mais oportunidades para obter revisões, apesar de não constarem mais dados ou literatura que apontem diretamente para a confirmação desta hipótese. No dito projeto, o *code review* parece ter provido impactos significativos, seja na qualidade geral do projeto ou diminuindo a inserção de novos defeitos e dívida técnica.

Assim, é possível concluir que os resultados deste trabalho corroboram com a hipótese apresentada, de que o *code review* está associado a impactos positivos na qualiade do código. Esta conclusão se dá pela averiguação de até 31% de redução em duplicação de código e até 71,66% em dívida técnica variável; é possível observar também a redução, ainda que em menor grau, de complexidade ciclomática difundida em três dos quatro projetos observados.

O APP1 sofreu um aumento de cerca de 37% na duplicação de código após a implantação do code review. Podemos tecer suposições que fatores relacionados ao planejamento estratégico do produto tiveram um impacto peculiar sobre este projeto, o que teria influenciado no aumento desta métrica. Apesar do expressivo resultado, não foram obtidos outros resultados que expliquem tal comportamento, nem literatura relacionada que embase uma possível relação deste resultado com a prática de *code review*.

6.2 *Survey* com os desenvolvedores

Ao todo, oito desenvolvedores participaram do experimento. Apesar do universo visivelmente limitado, incentivou-se a contribuição de todos, para almejando-se o maior apro-

veitamento possível. Além das perguntas específicas sobre a aplicação das técnicas e os objetivos alcançados descritos na seção 5.2, há questionamentos relacionados ao perfil dos envolvidos, para delimitar ainda mais nosso escopo de análise.

As questões elaboradas possuem 4 alternativas de resposta: duas expressando concordância (total ou parcial) e duas expressando discordância (total ou parcial). Estas sofreram pequenas alterações para melhor adequação semântica em algumas questões. São as questões:

- **Q1**: Há quantos anos você trabalha com desenvolvimento?
- **Q2**: Quantos anos você contribuiu até agora na organização atual?
- **Q3**: Você teve alguma experiência prévia com a prática de *code review*?
- **Q4**: Qual a sua principal área de desenvolvimento na empresa?
- **Q5**: O *code review* auxilou você a conhecer métodos, funções, classes e componentes dos sistemas que você não conhecia bem antes?
- **Q6**: Você se sente mais responsável pela qualidade geral do sistemas em que você atuou como revisor?
- **Q7**: Você recebeu comentários e observações pertinentes que ajudaram na qualidade do código que você escreveu?
- **Q8**: As discussões durante a revisão que você participou como autor ou revisor geraram resultados positivos, como a implementação de soluções melhores e aprendizado?
- **Q9**: Você sentiu algum tipo de problema social, como desentendimentos, rixas e inimizades geradas pelo *code review*?
- **Q10**: O *code review* gerou esforço extra considerável nas suas atividades diárias?
- **Q11**: O *code review* ocasionou atrasos no planejamento?
- **Q12**: Defeitos funcionais de código (*bugs*) foram encontrados e corrigidos antes da entrega nos *code reviews* que você participou, como revisor ou autor?

- **Q13**: Defeitos não funcionais de código (code smells) foram encontrados e corrigidos antes da entrega nos *code reviews* que você participou, como revisor ou autor?

O formulário foi disponibilizado para os desenvolvedores da equipe através da plataforma Google Forms[37], como mostra a figura 6.5.

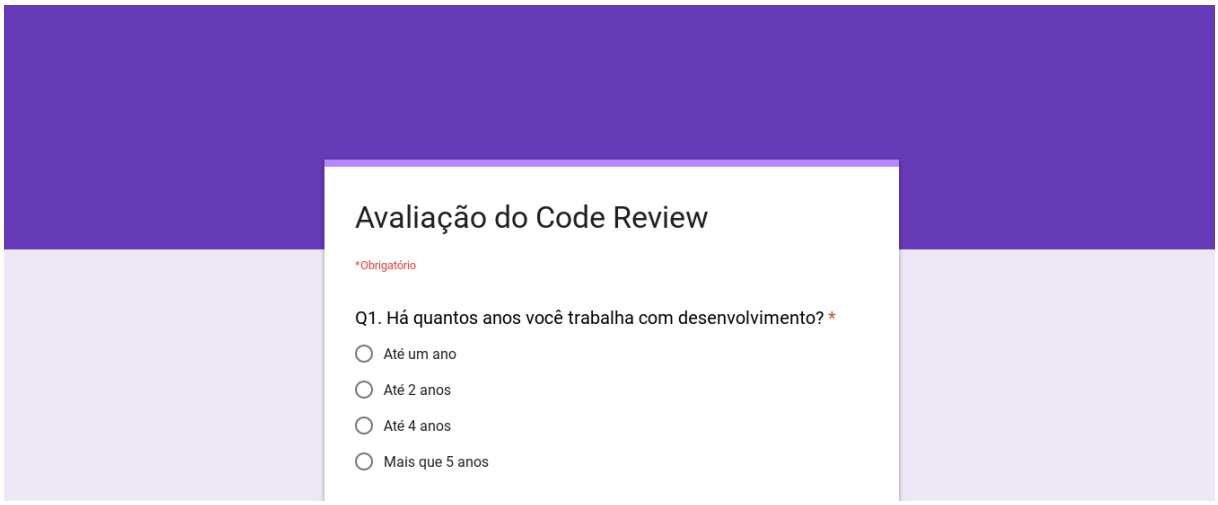

Figura 6.5: Demonstração do formulário utilizado na *survey*

Através dessa ferramenta é possível criar questões de múltipla escolha descritivas ou que representam uma escala, como era necessário para condução deste experimento.

As quatro primeiras questões nos ajudam a entender melhor o perfil dos envolvidos. Na Q1 observou-se que a maioria dos desenvolvedores atua na área há menos de 4 anos, enquanto a Q2 mostra que o todos os desenvolvedores se encontram na organização há menos de dois anos, como é disposto na figura 6.6

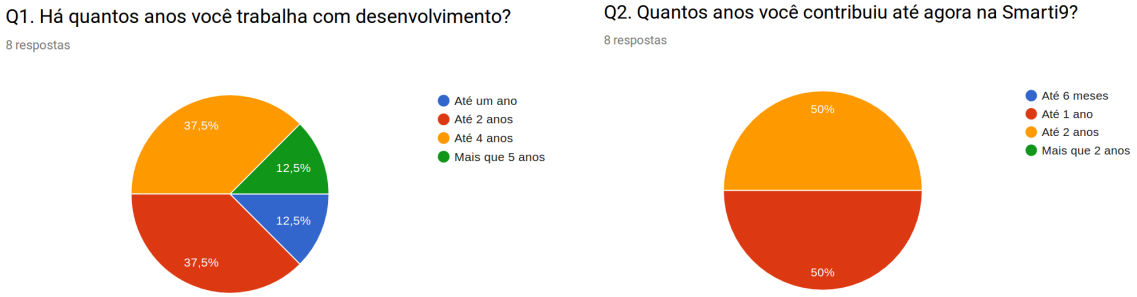

Figura 6.6: Resultados de Q1 e Q2

Enquanto a maioria não havia tido contato com o *code review*, alguns relataram

[37]https://www.google.com/forms/about/

experiências anteriores, como foi observado no Q3. Já Q4 indica que a maior parte do grupo é desenvolvedor Web/*Backend*, o que condiz com a distribuição da equipe estudada. Esses resultados podem ser vistos na figura 6.7.

Figura 6.7: Resultados de Q3 e Q4

O resultado de Q5, visível através da figura 6.8 mostra que 75% dos participantes não foi categórico em afirmar ou negar que o code review pode ajudar no conhecimento de componentes desconhecidos do software.

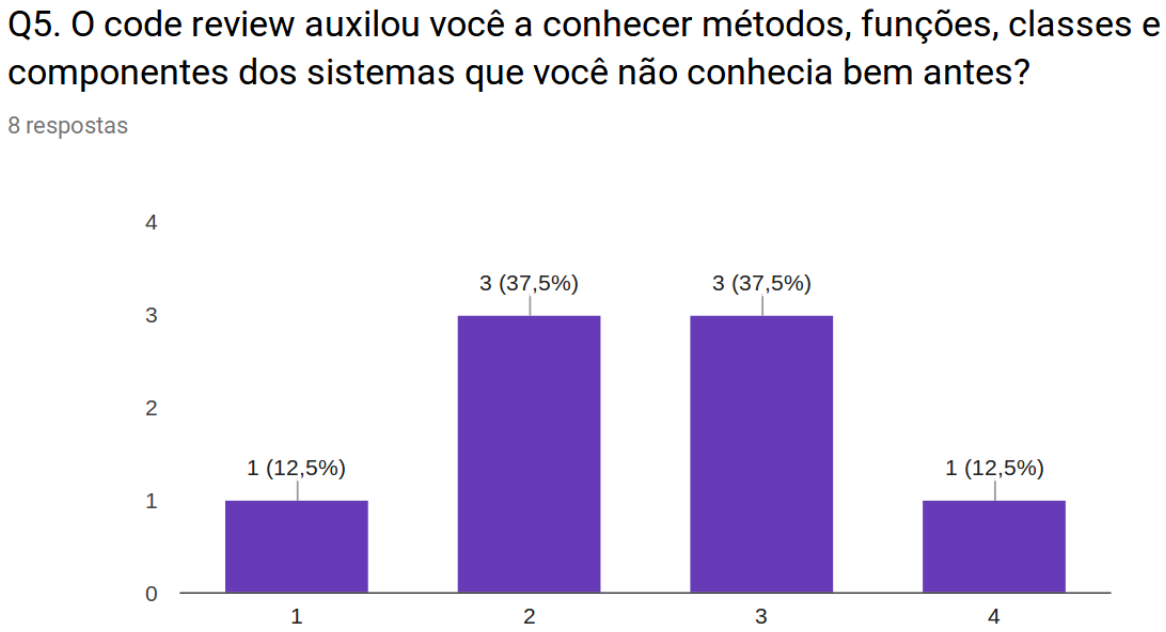

Figura 6.8: Resultados de Q5

Em Q6 pode-se observar que a maioria dos participantes acredita que, em certo grau, se sentiu mais responsável pelos projetos nos quais se tornou revisor (figura 6.9), enquanto a distribuição vista em Q7 (figura 6.10) mostra que grande parte dos desenvol-

vedores recebeu comentários e observações pertinentes durante o processo de revisão.

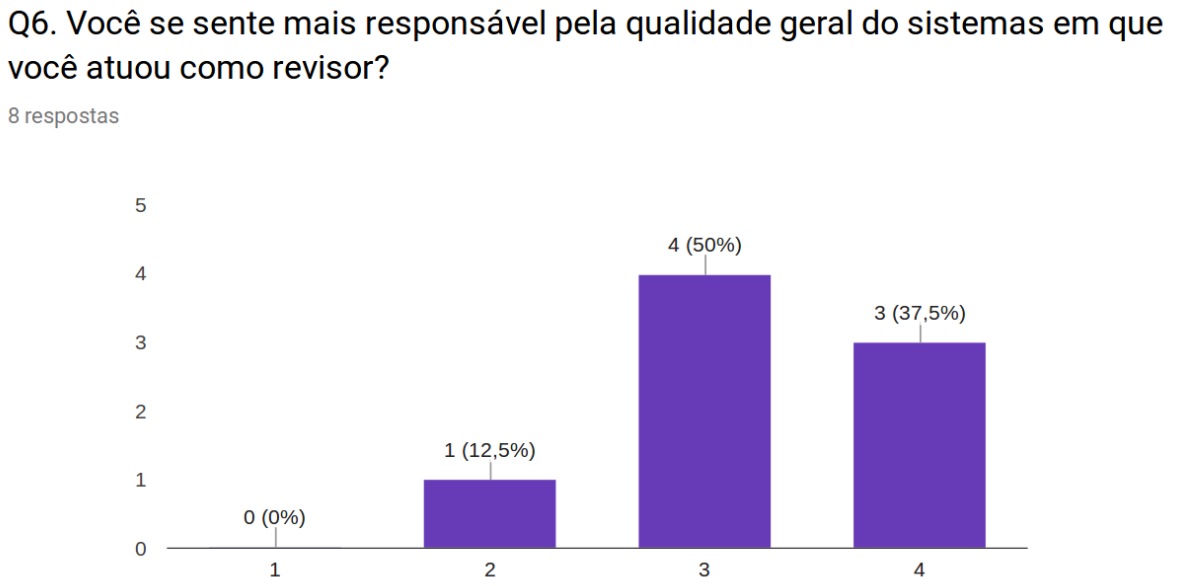

Figura 6.9: Resultados de Q6

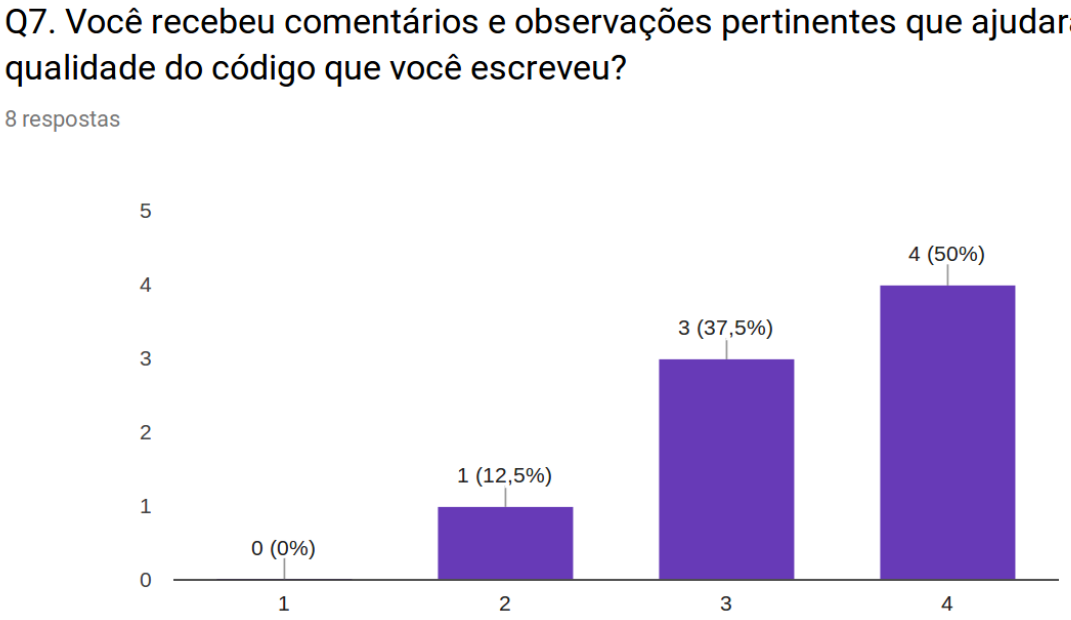

Figura 6.10: Resultados de Q7

Em Q8, 75% dos participantes parecem acreditar que as discussões ocorridas durante a prática geraram resultados positivos como implantação de melhores soluções e o aprendizado dos envolvidos, como pode ser visto na figura 6.11. Ao mesmo tempo, O

Q9 indica que absolutamente nenhum dos envolvidos, como mostra a figura 6.12, sentiu qualquer tipo de desentendimento ocasionado pela prática de revisão.

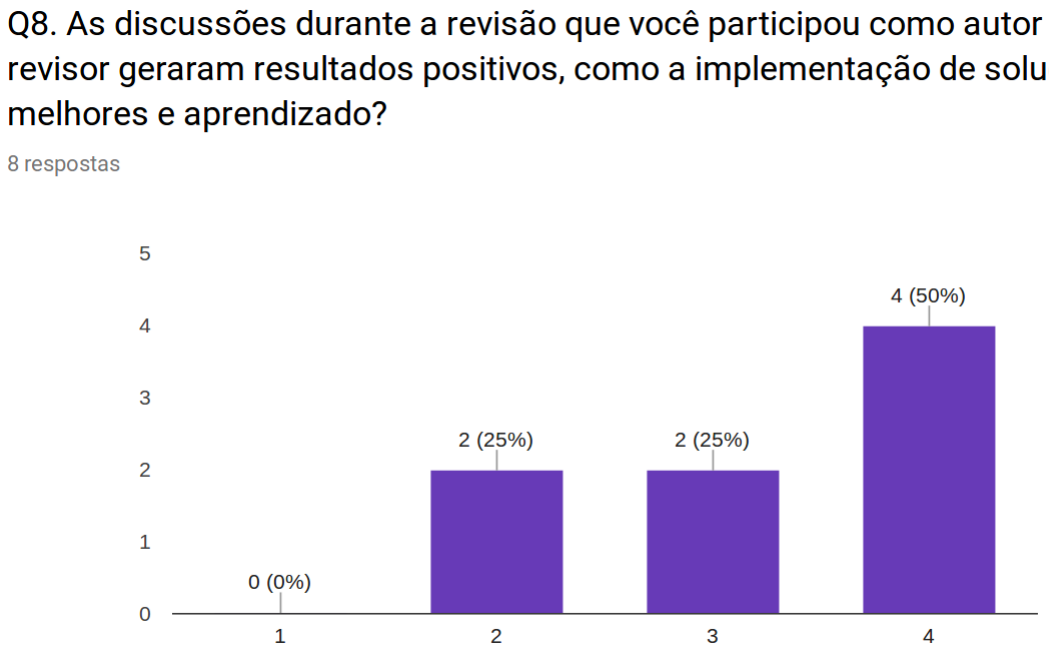

Figura 6.11: Resultados de Q8

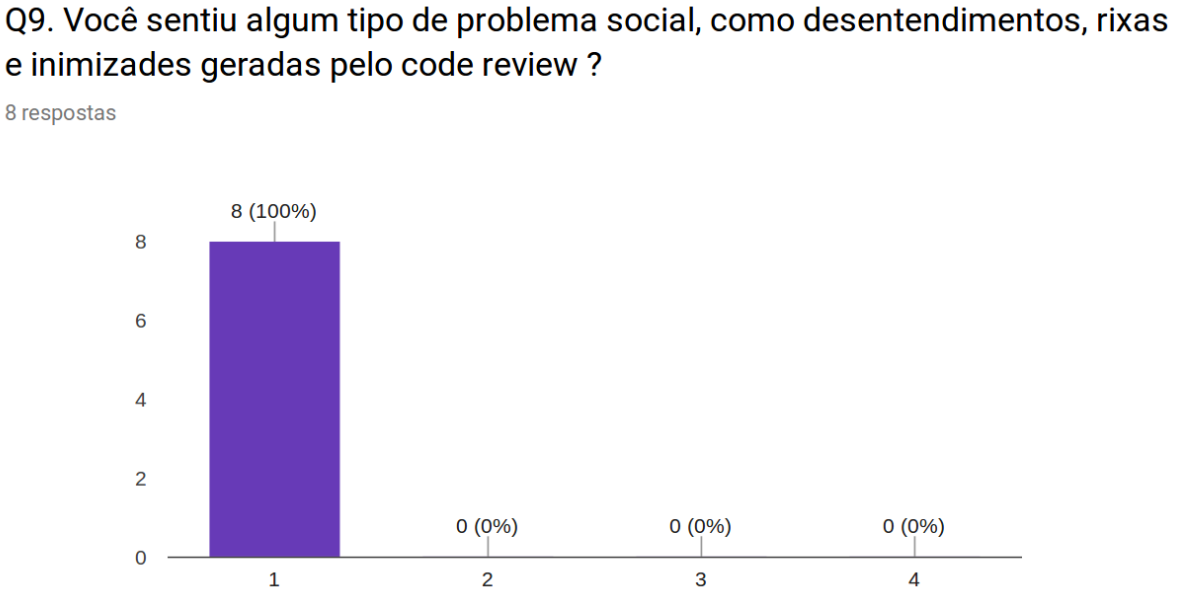

Figura 6.12: Resultados de Q9

Contudo, Q10 mostra que a maioria dos participantes acredita que a prática não adicionou esforço extra considerável em suas atividades diárias, apesar de ninguém afirmar categoricamente que não sentiu nenhum tipo de labor adicional em decorrência da nova

atividade. De maneira praticamente análoga, o Q11 indica que os desenvolvedores não associaram o *code review* à atrasos no planejamento. Esses resultados são arranjados nas figura 6.13 e 6.14, respectivamente.

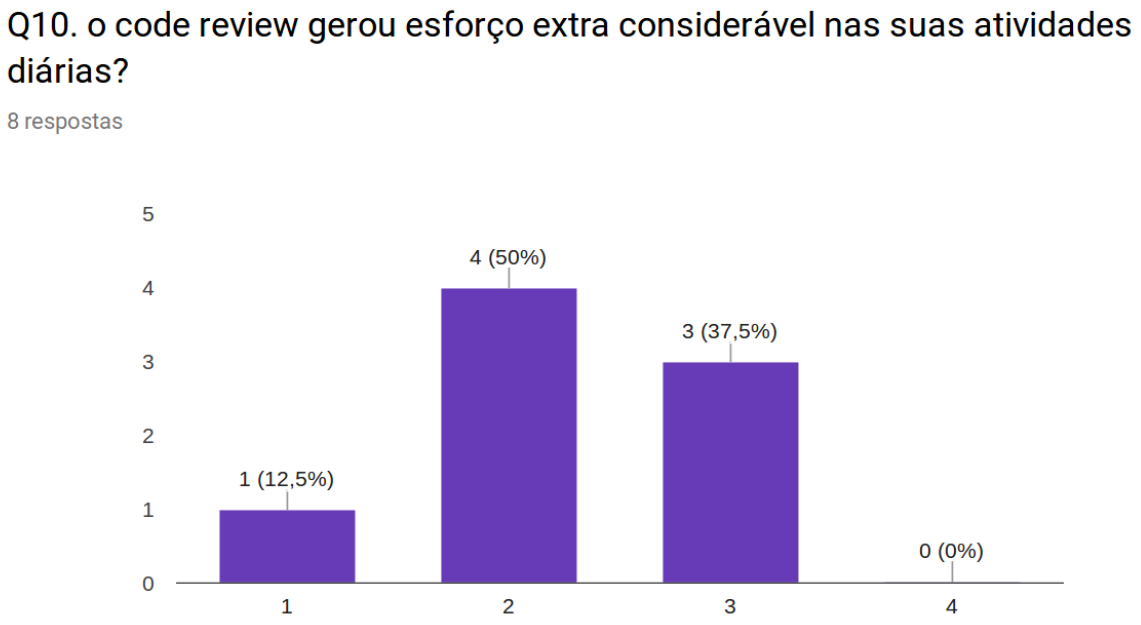

Figura 6.13: Resultados de Q10

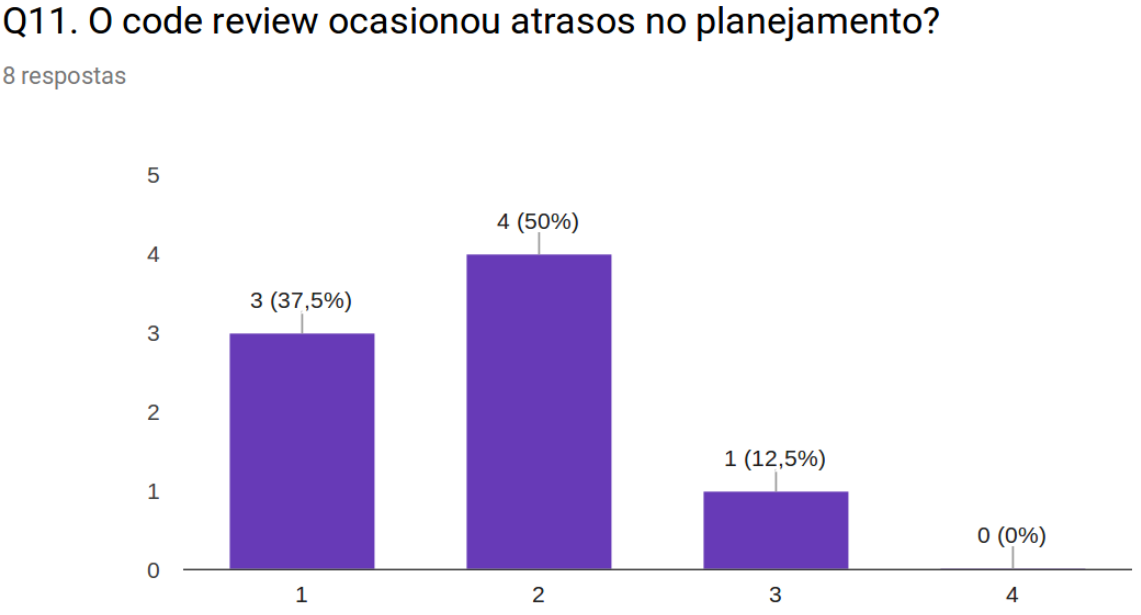

Figura 6.14: Resultados de Q11

Por fim, os resultados vistos em Q12, através da figura 6.15, mostram que 75% dos

colaboradores não encontrou ou recebeu indicações cotidianamente de defeitos funcionais durante a prática de *code review*. Já a figura 6.16 representa os resultados de Q13, onde 62,5% dos participantes perceberam a identificação e correção de defeitos estruturais (*code smells*) nas revisões que fizeram parte. Além disso, 50% declara que tal ocorrência era frequente, e nenhum participante externou que tal fato era raro.

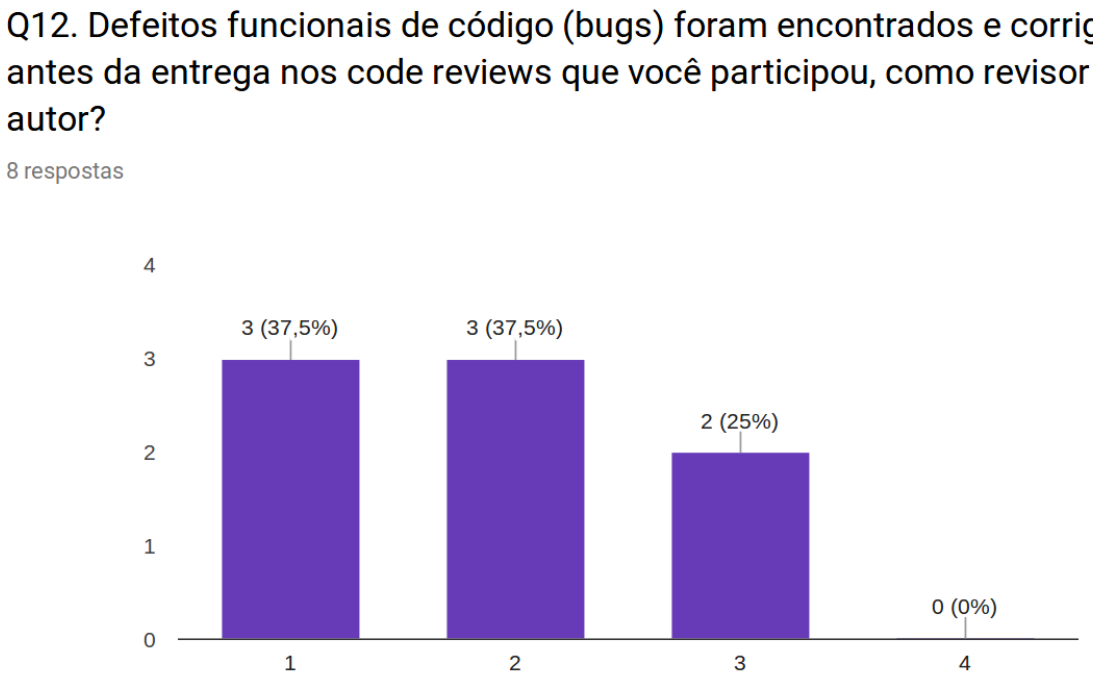

Figura 6.15: Resultados de Q12

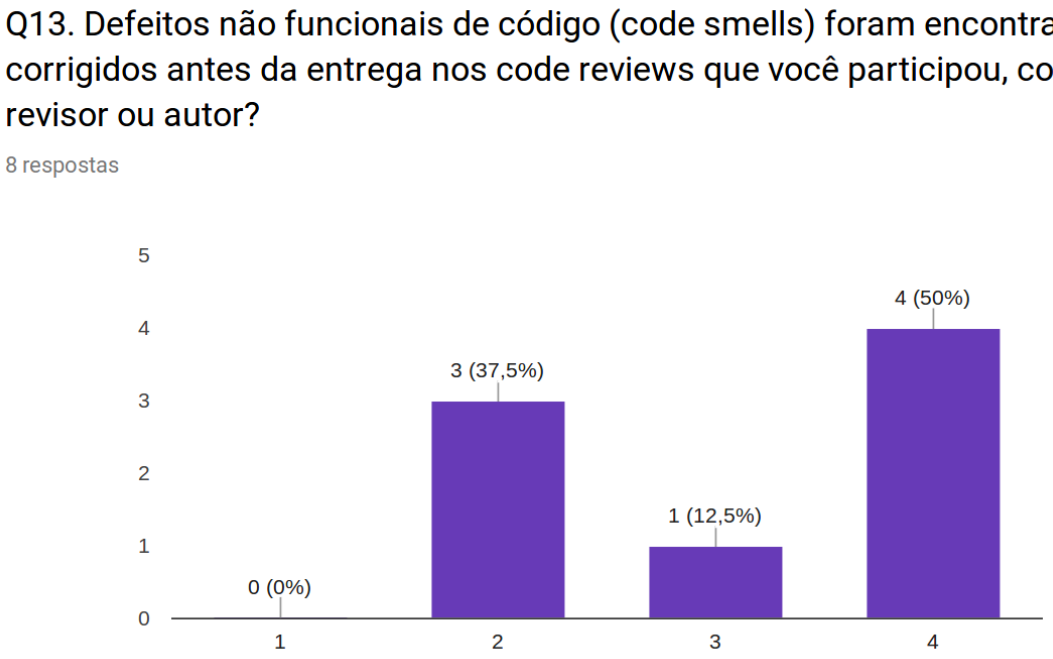

Figura 6.16: Resultados de Q13

Os resultados obtidos através da *survey* pode ser utilizada de *proxy* para análise da visão dos desenvolvedores, hora autores hora revisores, sobre prática e seu papel no processo de desenvolvimento. Estas métricas podem ser utilizadas para avaliação do alcance ou não dos objetivos propostas na seção 3.2.

Os resultados corroboram com a segunda hipótese levantada neste trabalho, que relaciona o *code review* a disseminação de conhecimento, debates e senso de responsabiliade compartilhada do código fonte. Através da *survey* foi possível identificar que 87,25% dos participantes se sentem mais responsáveis pelos projetos que revisou, enquanto 75% acreditam que implementaram soluções melhores e aprenderam com isso. É possível observar ainda que 87,25% acreditam ter recebido observações pertinentes.

O questionário aplicado aos participantes do experimento apontou que durante o *code review*, de maneira geral, foi possível realizar discussões pertinentes e disseminação de conhecimento. É possível observar que a maior parte dos desenvolvedores se sentiu mais responsável pelo sistemas que atuou como revisor, outro objetivo proposto neste trabalho. Sobre conhecer melhor os sistemas nos quais atuou, os participantes ficaram divididos, não gerando um consenso sobre o tema.

É possível afirmar que nenhum desenvolvedor declarou ter sentido desentendi-

mentos ou inimizades nutridas através da prática de *code review*. Este resultado pode estar relacionado ao processo de desenvolvimento adotado e ao perfil da organização e dos participantes do experimento.

Apesar de não haver unanimidade, é possível perceber que a tendência geral dos participantes em não associar o *code review* à expressivos esforços extras nem como causa fundamental de atrasos do planejamento.

Por fim, através do questionário foi possível obter outros resultados congruentes com a literatura: A maior parte dos desenvolvedores não notou, rotineiramente, defeitos funcionais de código sendo corrigidos através das revisões que participou, enquanto a maioria vivenciou o contrário com defeitos não funcionais de código (estruturais) nesse mesmo contexto (BELLER et al., 2014).

7 Considerações Finais

O *code review* está presente na fabricação de software há mais de 30 anos, sempre adaptando-se para acompanhar a evolução dos modelos de desenvolvimento na indústria. As tradicionais reuniões de revisão vêm sendo substituídas por práticas mais flexíveis, apoiadas pela automação e muitas vezes associadas à metodologias ágeis de desenvolvimento. O surgimento de ferramentas computacionais que embasam esta prática fomenta sua incorporação em processos de desenvolvimento de diferentes portes e contextos.

A prática é amplamente associada à impactos positivos na qualidade de software e na disseminação de conhecimento. Em alguns cenários são vislumbrados impactos ainda mais específicos, como a detecção e correção de vulnerabilidades ou no crescimento da participação dos membros da comunidade *Open Source*.

A literatura aborda alguns aspectos negativos tradicionalmente relacionados à prática. O mais consoante, também corroborado pelo senso comum, é o exforço extra e o inerente aumento do custo do processo, uma vez que a atividade de revisão é essencialmente realizada por um ser humano.

Finalmente, os prazos curtos e o apuro por entregas, conhecidos componentes do processo de desenvolvimento, podem gerar atividades de revisão sem o devido cuidado e zelo, aumentando a probabiliade de revisões inócuas e de pouco valor, diminuindo o custo/benefício da prática.

Existem ameaças à validade dos resultados aqui obtidos, que devem ser cuidadosamente observadas e ponderadas para utilização destes dados em aplicações industriais ou como base para trabalhos futuros. A maior parte destas ressalvas está relacionada ao ambiente real no qual foi conduzido o experimento, como qualquer trabalho de Engenharia de Software experimental (SJOBERG et al., 2002).

Apesar dos os esforços contrários relatados neste trabalho, os resultados podem ter sido influenciados, positiva ou negativamente, por variáveis alheias ao experimento, não mapeadas e relacionadas ao contexto da organização e dos projetos, como escopo, planejamento e distribuição de recursos. Nesta seção iremos discutir algumas delas.

As ameaças à generalização dos resultados aqui obtidos são caracterizadas como ameaças externas (WOHLIN et al., 2012). Nesta categoria podemos elencar o modelo de desenvolvimento ágil e o enquadramento de *startup* desempenhados pela organização como ameaças externas, uma vez que tais características estreitam o leque de situações onde os resultados aqui descritos podem ser considerados válidos.

Wohlin et al. (2012) descrevem os riscos das influência de variáveis não controladas e desconhecidas como ameaças internas a validade dos resultados. Neste aspecto podemos destacar a pequena quantidade de participantes, o que deixa os resultados mais sensíveis a variação de produtividade dos desenvolvedores por motivos alheios ao escopo do trabalho. Além disso o acréscimo da maturidade, sentimento de avaliação do processo e preocupação com a qualidade de código geradas pela pesquisa podem impactar os resultados apresentados.

Representando tanto ameças internas quanto externas, está a distribuição dos objetos de pesquisa; o fato dos projetos estudados estarem ligados à mesma organização e equipe pode enviesar os resultados pode impactar nos resultados, uma vez que objetivos estratégicos da organização podem influenciar na qualidade do código, independentemente do *code review*.

Existem diversas formas de extender este trabalho e almejar resultados mais consistentes. Como trabalhos futuros, é proposto um aumento no tempo de observação e agregar novos projetos e organizações ao experimento. Além disso, inserir novas métricas de qualidade de software e especialmente ligadas à execução do *code review*, como quantidade de observações, efetividade das revisões e interatividade dos participantes pode enriquecer a discussão e levar a novos resultados.

Finalmente, é sugerido utilizar os dados deste e de futuros experimentos para reavaliar as decisões tangentes ao processo adotado, realizando possíveis alterações na proposta e comparar suas diferentes versões.

Referências Bibliográficas

ACKERMAN, A. F.; FOWLER, P. J.; EBENAU, R. G. Software inspections and the industrial production of software. In: *Proc. Of a Symposium on Software Validation: Inspection-testing-verification-alternatives*. [S.l.: s.n.], 1984. p. 13–40. ISBN 0-444-87593-X.

AMAN, H. 0-1 programming model-based method for planning code review using bug fix history. In: . [S.l.: s.n.], 2013. v. 2, p. 37–42.

ARAúJO, M. A. P. et al. Métodos estatísticos aplicados em engenharia de software experimental. In: SOFTWARE, X. S. B. de Engenharia de (Ed.). *ANAIS do XX SBBD*. [S.l.], 2006. p. 325.

ASUNDI, J.; JAYANT, R. Patch review processes in open source software development communities: A comparative case study. In: *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*. [S.l.: s.n.], 2007. p. 166c–166c. ISSN 1530-1605.

AURUM, A.; PETERSSON, H.; WOHLIN, C. State-of-the-art: software inspections after 25 years. *Softw. Test., Verif. Reliab.*, v. 12, n. 3, p. 133–154, 2002.

BACCHELLI, A.; BIRD, C. Expectations, outcomes, and challenges of modern code review. In: *Proceedings of the 2013 International Conference on Software Engineering*. [S.l.]: IEEE Press, 2013. (ICSE '13), p. 712–721. ISBN 978-1-4673-3076-3.

BASILI, V. R.; WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Trans. Softw. Eng*, SE-10, no. 6, p. 728–738, 1984.

BAVOTA, G.; RUSSO, B. Four eyes are better than two: On the impact of code reviews on software quality. In: . [S.l.: s.n.], 2015. p. 81–90.

BAYSAL, O. et al. The secret life of patches: A firefox case study. In: *2012 19th Working Conference on Reverse Engineering*. [S.l.: s.n.], 2012. p. 447–455. ISSN 1095-1350.

BAYSAL, O. et al. The influence of non-technical factors on code review. In: . [S.l.: s.n.], 2013. p. 122–131.

BELLER, M. et al. Modern code reviews in open-source projects: Which problems do they fix? In: . [S.l.: s.n.], 2014. p. 202–211.

BERNHART, M. et al. A task-based code review process and tool to comply with the do-278/ed-109 standard for air traffic managment software development - an industrial case study. In: . [S.l.: s.n.], 2011. p. 182–187.

BOEHM, B.; BASILI, V. R. Software defect reduction top 10 list. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 34, n. 1, p. 135–137, 12 2001. ISSN 0018-9162.

BOSU, A.; CARVER, J. Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In: . [S.l.: s.n.], 2014.

DYBÅ, T.; KAMPENES, V. B.; SJØBERG, D. I. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, Elsevier, v. 48, n. 8, p. 745–755, 2006.

FAGAN, M. E. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 15, n. 3, p. 182–211, 09 1976. ISSN 0018-8670.

FOWLER, M. et al. *Refactoring: Improving the Design of Existing Code*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0-201-48567-2.

FROST, J. *Using Hypothesis Tests to Bust Myths about the Battle of the Sexes*. 2013. [Online; Acessado em 12/06/2017]. Disponível em: <http://blog.minitab.com/blog/adventures-in-statistics-2/using-hyp\ \othesis-tests-to-bust-myths-about-the-battle-of-the-sexes>.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

KEMERER, C. F.; PAULK, M. C. The impact of design and code reviews on software quality: An empirical study based on psp data. *IEEE Transactions on Software Engineering*, v. 35, n. 4, p. 534–550, 07 2009. ISSN 0098-5589.

KITCHEHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. *Tech. report, Ver. 2.3 EBSE*, v. 9, n. 9, p. 9, 2007.

KONONENKO, O. et al. Investigating code review quality: Do people and participation matter? In: . [S.l.: s.n.], 2015. p. 111–120.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, I. Technical debt: From metaphor to theory and practice. *Ieee software*, IEEE, v. 29, n. 6, p. 18–21, 2012.

LETOUZEY, J.-L. The sqale method for evaluating technical debt. In: IEEE. *Managing Technical Debt (MTD), 2012 Third International Workshop on*. [S.l.], 2012. p. 31–36.

MCINTOSH, S. et al. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: . [S.l.: s.n.], 2014. p. 192–201.

MENEELY, A. et al. An empirical investigation of socio-technical code review metrics and security vulnerabilities. In: . [S.l.: s.n.], 2014. p. 37–44.

MILLER, J. et al. Statistical power and its subcomponents—missing and misunderstood concepts in empirical software engineering research. *Information and Software Technology*, Elsevier, v. 39, n. 4, p. 285–295, 1997.

MISHRA, R.; SUREKA, A. Mining peer code review system for computing effort and contribution metrics for patch reviewers. In: . [S.l.: s.n.], 2014. p. 11–15.

MORALES, R.; MCINTOSH, S.; KHOMH, F. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 171–180, 2015.

Pennsylvania State University. *Lesson 9: Comparing Two Groups*. 2017. [Online; Acessado em 12/06/2017]. Disponível em: <https://onlinecourses.science.psu.edu/stat200/book/export/html/57>.

Pennsylvania State University. *Tests for Error Normality*. 2017. [Online; Acessado em 12/06/2017]. Disponível em: <https://onlinecourses.science.psu.edu/stat501/node/366>.

PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide*. [S.l.]: John Wiley and Sons, 2008.

PRAUSE, C. R.; APELT, S. An approach for continuous inspection of source code. In: ACM. *Proceedings of the 6th international workshop on Software quality*. [S.l.], 2008. p. 17–22.

RAYMOND, E. S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. [S.l.]: O'Reilly and Associates, Inc., 2001. ISBN 0-596-01008-8.

SJOBERG, D. I. K. et al. Conducting realistic experiments in software engineering. In: *Proceedings International Symposium on Empirical Software Engineering*. [S.l.: s.n.], 2002. p. 17–26.

THONGTANUNAM, P. et al. Improving code review effectiveness through reviewer recommendations. In: . [S.l.: s.n.], 2014. p. 119–122.

THONGTANUNAM, P. et al. Investigating code review practices in defective files: An empirical study of the qt system. In: . [S.l.: s.n.], 2015. v. 2015, p. 168–179.

THONGTANUNAM, P. et al. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In: . [S.l.: s.n.], 2015. p. 141–150.

VOTTA, J.; LAWRENCE, G. Does every inspection need a meeting? *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 18, n. 5, p. 107–114, 12 1993. ISSN 0163-5948.

WANG, J. *Using MINITAB*. 2009. [Online; Acessado em 12/06/2017]. Disponível em: <http://www.stat.wmich.edu/wang/664/egs/\\MTBrust.html>.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

XIA, X. et al. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*. [S.l.: s.n.], 2015. p. 261–270.