Avaliação de Cenários de Uso da Programação em Pares por Experiência de Desenvolvedores de Software, Trabalho em Progresso e Complexidade de Tarefas

Fernando Jean Silva Rocha[1], **Lesandro Ponciano**[1]

[1]Bacharelado em Engenharia de Software – PUC Minas
Ed. Fernanda. Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

fernandojean8@gmail.com, lesandrop@pucminas.br

Abstract. *Pair programming is a software development practice in which two programmers implement the software together, on the same workstation. This practice was defined in the Extreme Programming (XP) software development process in order to obtain quality and speed in software coding. Although well studied, little is known about how this practice is used outside the XP process and in conjunction with current software development approaches. In this study, we seek to assess, in an experimental approach, which usage scenarios make the application of pair programming effective. The study consists of remote pair programming. In the scenarios evaluated, we characterize the effect of the following factors: work in progress, complexity of tasks, and developer experience. To that end, 23 people participated in an experiment. The results show that the scenario of using pair programming has an effect on the complexity of the generated code, development time and code maintenance rates.*
Keywords: *Pair Programming, development time, Complexity, Maintainability*

Resumo. *Programação em pares é uma prática de desenvolvimento de software em que dois programadores codificam o software juntos, em uma mesma estação de trabalho. Essa prática foi definida no processo de desenvolvimento de software Extreme Programming (XP), visando obter qualidade e velocidade na produção de código. Apesar de bastante estudada, pouco se sabe sobre como essa prática é usada fora do processo XP e em conjunto com a abordagens atuais de desenvolvimento de software. Neste estudo, busca-se avaliar, de forma experimental, quais cenários de uso tornam adequada a aplicação da programação em pares. O estudo consiste na programação em pares realizada de forma remota. Dentre os cenários avaliados, busca-se caracterizar o efeito dos fatores seguintes: trabalho em progresso, complexidade de tarefas e experiência dos desenvolvedores. Para esse fim, 23 pessoas participaram de um experimento. Os resultados apontam que os cenários de uso de programação em pares têm efeito sobre o complexidade do código gerado, tempo de desenvolvimento e índices de manutenibilidade do código.*
Palavras-Chave: *Programação em Pares, tempo de desenvolvimento, Complexidade, Manutenibilidade*

1. Introdução

A metodologia escolhida para o desenvolvimento é um fator determinante para o sucesso ou fracasso do projeto de *software*. O mau gerenciamento dos riscos potenciais

pode afetar a produtividade, o custo, o prazo e a qualidade, determinando o fracasso do *software* final [Chaouch et al. 2019]. Em consequência disso, as metodologias ágeis como o *Scrum*, a Programação Extrema (XP, do inglês *Extreme Programming*) e o Desenvolvimento Dirigido por Funcionalidades (FDD, do inglês *Feature Driven Development*) buscam amenizar os riscos por meio de práticas implícitas, como ciclos de entrega pequenos, *feedbacks* constantes e planejamento incremental [Beck 2000, Newkirk 2002, Anwer et al. 2017, Chaouch et al. 2019].

O processo XP é utilizado em projetos complexos. Trata-se de um processo ágil, com pouca documentação, disciplinado no estabelecimento de práticas e flexível na sua adaptação a diferentes contexto. Ele permite realizar o gerenciamento de mudanças rapidamente com poucos riscos para o projeto [Newkirk 2002]. XP possui um conjunto de regras e práticas que são empregadas nas seguintes atividades: planejamento, projeto, codificação e testes [Pressman 2011]. Dentre as práticas que são aplicadas em cada uma dessas atividades, a que é estudada neste trabalho é a programação em pares (do inglês *Pair Programming*), que está na etapa de codificação.

A programação em pares é uma prática relevante na atividade de codificação [Pressman 2011]. Essa prática consiste em dois programadores trabalharem juntos na mesma máquina [Beck 2000] para gerar um software com alta qualidade, baixo risco e menor custo [Anwer et al. 2017]. Nesse sentido, a premissa da programação em pares é que parte dos defeitos são identificados e corrigidos por um dos programadores durante a codificação. Apesar da programação em pares ser amplamente utilizada por praticantes da metodologia XP, pouco se sabe sobre o emprego e efeito dessa prática em diferentes cenários fora desta metodologia. Além disso, com atual avanço das práticas de programação remota, em que os programadores se encontram dispersos geograficamente [Xinogalos et al. 2019, Saltz and Heckman 2020], mostra-se relevante compreender o emprego de programação em pares nesse contexto.

Nesse contexto, a questão de pesquisa que este trabalho busca responder, verificando quais aspectos fundamentam a aplicação deste tipo de desenvolvimento é: Quais os melhores cenários para a aplicação da programação em pares? Ao buscar uma resposta para essa questão, o estudo investiga a eficiência da programação em pares em contextos de desenvolvimento de *softwares* em que se varia o trabalho em progresso (WIP, do inglês *Working in Progress*), experiência dos desenvolvedores e a complexidade das tarefas implementadas pelos desenvolvedores.

O objetivo deste trabalho é avaliar cenários de uso da programação em pares levando em consideração o WIP, a experiência do desenvolvedor e a complexidade da tarefa para identificar o cenário mais adequado para a aplicação dessa prática. Avalia-se a adequação dessa prática por meio do tempo de desenvolvimento, manutenibilidade e complexidade do software gerado. Para atingir esse objetivo são definidos os seguintes objetivos específicos:

1. Caracterizar o efeito do WIP na programação em pares;
2. Caracterizar o efeito da complexidade da tarefa a ser implementada na programação em pares;
3. Caracterizar o efeito da experiência do desenvolvedor na programação em pares;
4. Verificar a adequação da prática de programação em pares.

O estudo é conduzido com 23 desenvolvedores de software de diferentes níveis de experiência. Eles atuam em um experimento em que se varia a complexidade da tarefa e o tamanho do WIP. Os resultados obtidos mostram que a programação em pares tem efeito no tempo de desenvolvimento, na manutenibilidade e complexidade do código produzido. Desenvolvedores experientes que atuam de forma individual tendem a gerar softwares com maior índice de manutenibilidade, enquanto observa-se maior índice de complexidade e tempo de desenvolvimento de software produzido por pares de programadores que combina programadores experientes e programadores não experientes. O restante deste texto está estruturado em outras 6 seções. A seção 2 apresenta a fundamentação teórica que norteia o trabalho. Os trabalhos relacionados são discutidos na Seção 3. Em seguida, a seção 4 apresenta a metodologia a ser utilizada no trabalho, destacando os materiais e métodos de avaliação. A seção 5 apresenta os resultados obtidos com a realização do experimento. Por fim, são apresentadas as discussões dos resultados (Seção 6) e as conclusões e trabalhos futuros (Seção 7).

2. Fundamentação Teórica

Esta seção de fundamentação teórica aborda os tópicos de programação em Pares, complexidade de *software*, qualidade de *software*, e trabalho em progresso. Esses tópicos são analisados a seguir.

2.1. Programação em Pares

A programação em pares é uma prática utilizada no processo XP, no qual dois programadores trabalham juntos em uma mesma máquina [Beck 2000] e em um mesmo código, com o objetivo de produzir um código de alta qualidade em um menor tempo [Nagappan et al. 2003]. Durante o desenvolvimento, um dos programadores atua como um piloto, produzindo artefatos como o código, enquanto o outro atua como um copiloto, assistindo, auxiliando e buscando abordagens alternativas para que se produza um código de melhor qualidade [Anwer et al. 2017].

Os programadores realizam um revezamento das funções em um tempo estipulado pela dupla [Nawrocki and Wojciechowski 2001], a fim de evitar o engessamento do tempo. A troca de funções permite evitar o cansaço dos programadores, além de manter a dupla engajada [Nagappan et al. 2003]. Ao se realizar a troca, o programador piloto contextualiza o outro para que seja possível validar o entendimento e dar continuidade no trabalho por meio da troca de conhecimentos[Man and Chan Keith 2006].

Na programação em pares, as habilidades comunicativas são fundamentais para a transferência de conhecimento entre os membros e na confiança para o desenvolvimento com o parceiro [Stadler et al. 2019]. Além disso, a confiança nas próprias habilidades e nas do parceiro de desenvolvimento tem influência significativa na compatibilidade e interação entre eles [Tsompanoudi et al. 2019] de modo que pode-se afetar positivamente ou negativamente o desenvolvimento. Outros fatores que devem ser levados em consideração é a auto-estima, estilo de aprendizado e a distribuição desigual da carga de trabalho entre os membros do par [Xinogalos et al. 2019].

A programação em pares distribuída é uma prática derivada da programação em pares. Na programação em pares distribuída, os programadores dispersos geograficamente [Stadler et al. 2019] e fazem o desenvolvimento do *software* de forma remota por

meio de uma infraestrutura adequada e especializada [Tsompanoudi et al. 2019]. O desenvolvimento distribuído utiliza *softwares* que permitem os membros do par se comunicarem, com o objetivo de coordenar o desenvolvimento e escrever o código utilizando um repositório ou editor compartilhado [Schümmer and Lukosch 2009].

2.2. Qualidade e Complexidade de Software

Como um dos objetivos deste trabalho é investigar o impacto que a abordagem de programação em pares pode ter na qualidade do *software*, mostra-se relevante analisar este conceito. Portanto, a qualidade de *software* é o grau de qualidade que o *software* possui, levando em consideração um conjunto de características e atributos desejados [IEEE 1993]. A norma ISO/IEC 25010 conceitua um conjunto de características de qualidade [ISO/IEC 2010]. Tais características são : Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade. Dentre as características citadas, este estudo analisa os aspectos de Manutenibilidade, Usabilidade e Funcionalidade.

Para cada uma dessas características, existe um conjunto de atributos que determina a qualidade do *software* [ISO/IEC 2010]. Existem dois fatores definidos pela norma ISO/IEC 25010 que podem afetar a qualidade do *software*. O primeiro fator está relacionado aos atributos de qualidade interna, como a quantidade de linhas de código. Já o segundo fator está relacionado aos atributos internos, como a adaptabilidade [Alshayeb 2009].

A complexidade de *software* é um fator interno. Ela pode ser entendida de forma diferente de acordo com o contexto. Uma das formas utilizadas para calcular a complexidade é por meio da métrica: Complexidade Ciclomática. Ela calcula o número de caminhos linearmente independentes da execução de um programa [McCabe 1976]. Entretanto, essa métrica desconsidera os fatores humanos como os comentários, a estética do código e as nomeações de classes, métodos e variáveis [Kearney et al. 1993]. Portanto, para realizar esta análise deve-se levar em consideração os fatores que podem afetar a complexidade como o tamanho, elementos sintáticos e semânticos, fluxos, nomes, organização e regras de negócio [Ajami et al. 2017].

2.3. WIP e a Importância de Limitá-lo

Como mencionado anteriormente, WIP é o trabalho em progresso, onde representa o número de itens de tarefas que uma pessoa ou time possui quando está trabalhando em um determinado momento. Ele serve para demonstrar a capacidade do fluxo de trabalho da equipe [Sereno et al. 2011]. O limite do WIP serve para restringir a quantidade de tarefas nas diferentes etapas de um fluxo de trabalho, a fim de que se consiga atuar somente nas tarefas que estão ativas.

O excesso de WIP pode fazer com que o processo fique ineficiente, pois deve-se inicializar uma nova atividade somente quando a atual estiver concluída. Com isso, o processo flui de forma uniforme e contínua. Quando isso não acontece, ocorrem frequentes trocas de contextos, de modo que os programadores necessitam ficar relembrando o que estavam fazendo para dar continuidade ao trabalho, afetando assim a produtividade [Wang et al. 2012]. Portanto, a limitação do WIP ajuda a evitar gargalos nos processos, restringe o trabalho em andamento, permitindo gerenciar melhor o fluxo de tarefas para evitar sobrecargas.

2.4. Código Coletivo

A programação em pares faz com que dois programadores atuem em conjunto em uma mesma solução, de modo que ambos fazem o compartilhamento de ideias e colaboram um com o outro durante o desenvolvimento. A prática da programação em pares leva a uma outra prática do XP denominada de código coletivo. Essa prática permite que cada membro da equipe possa trabalhar em qualquer parte do código seja qual for o momento. Dessa forma, caso um programador encontre algo que possa ser melhorado não é necessário notificar a pessoa que desenvolveu esse trecho de código para que ela faça a correção, pois o programador que encontrou pode aplicar a refatoração devido ao seu conhecimento obtido. Isso acontece pois ao se trabalhar com a programação em pares, ocorre o compartilhamento de conhecimento [Qureshi and Ikram 2015].

3. Trabalhos Relacionados

Nesta seção, são apresentados trabalhos em que foram realizadas pesquisas semelhantes, correlacionadas ou que apresentam informações relevantes para a abordagem proposta neste estudo. Os trabalhos relacionados discutidos nesta seção envolvem em grande parte estudos realizados sobre a programação em pares, considerando a qualidade do *software* desenvolvido, a complexidade, o tempo e a experiência dos programadores.

Segundo Beck (2000), a programação em pares ajuda a garantir que determinadas práticas, como a refatoração, sejam sempre realizadas, uma vez que mesmo que o programador não queira fazê-las o seu parceiro poderá. Dessa forma, é possível evitar o estresse do programador durante o desenvolvimento, garantir a qualidade do código produzido pela equipe e as chances de se ignorar o compromisso com as outras equipes se torna menor do que quando se está trabalhando sozinho [Beck 2000]. As principais contribuições deste estudo para esse trabalho são: a identificação de fatores que são melhorados quando o processo de desenvolvimento é realizado em dupla como o aumento da taxa de refatorações de código contribuindo para um código de melhor qualidade.

Em Arisholm et al. (2007), foi realizado um estudo sobre a programação em pares levando em consideração a complexidade do sistema e a experiência dos programadores. No experimento, foram realizadas várias tarefas com diferentes graus de complexidade. Como resultado, foi constatado que a programação em pares reduz o tempo de desenvolvimento de uma tarefa. Dessa forma, o estudo afirma que houve um aumento significativo de esforço para se produzir um código de forma correta e de melhor qualidade, enquanto no sistema mais complexo houve um aumento na geração de soluções corretas. Já nos *softwares* menos complexos, houve uma redução no tempo gasto, mas não houve diferença significativa quanto à solução gerada. O estudo se mostra relevante para este trabalho, pois apresenta fatores que devem ser analisados quando se trabalha com a programação em pares. Dentre os fatores apresentados, destaca-se a experiência do desenvolvedor e a complexidade da tarefa.

Em seu estudo, Hannay et al. (2009) verificaram a eficácia da programação em pares por meio de uma meta-análise de 18 estudos. O estudo mostra que houve um aumento significativo na qualidade e uma diminuição na duração da realização de tarefas com a programação em pares. Além disso, observou-se que a programação em pares possui um desempenho melhor que a programação individual quando a complexidade da atividade realizada é baixa. Já nas complexas, o código gerado apresenta uma maior qua-

lidade [Hannay et al. 2009]. Esse estudo foi conduzido em 2009. Ele reforça os objetivos deste trabalho de se analisar a programação em pares nos contextos atuais de desenvolvimento de software.

Coman et al. (2014) realizam um estudo com o objetivo de identificar situações em que a programação em pares é realmente necessária e benéfica. Para isso, eles selecionaram dois diferentes projetos em termos de tamanho e tempo, estudaram e classificaram os diferentes tipos de interações entre os programadores avaliando a finalidade e a ocorrência de cada interação [Coman et al. 2014]. Após a análise, identificaram que as interações tiveram duas características importantes:

1. A programação solo é um componente importante em determinados problemas;
2. As interações possuem dois objetivos principais que são: o auxiliar um membro da equipe que esteja com dificuldades e a colaboração entre membros que compartilham o mesmo objetivo na solução de um problema.

Como conclusão do estudo, validaram que as interações entre os membros de cada par são semelhantes às que ocorrem quando se aplica a programação em pares. Além disso, foi constatado que os desenvolvedores que trabalham sozinhos reconheceram que há benefícios em realizar essa prática, pois o problemas que podem ser resolvidos de maneira individual são mais simples, enquanto os mais complexos necessitam do auxílio de outras pessoas. Entretanto, a programação obrigatória em pares, quando pode ser realizada de maneira individual, é menos eficiente [Coman et al. 2014]. O trabalhos deles contribui para o presente estudo ao mostrar que o tipo de interação realizada entre os membros é um fator que deve ser levado em consideração quando se está analisando a programação em pares.

Ao realizar um estudo comparativo entre as práticas do *Scrum* e XP, Anwer et al. (2017) apresentam as características, fases, práticas, valores e papéis dessas metodologias com o objetivo de identificar as semelhanças e diferenças entre elas [Anwer et al. 2017]. O valor do respeito que um programador deve ter ao outro apresentado no trabalho, é um dos pilares para permitir implementar práticas como a programação em pares. Os desenvolvedores devem aprender a respeitar o próximo levando em consideração as suas crenças, dificuldades de aprendizagem, habilidades comunicativas e a confiança no parceiro [Stadler et al. 2019]. Dessa forma, esse valor é importante na formação de pares para que o resultados do trabalho cooperativo seja efetivo.

No estudo sobre as percepções e experiências relacionadas a programação em pares feito por Tsompanoudi et al. (2019), constatou que o membro da equipe com a maior experiência ou maior autoconfiança em programação tende a conduzir a implementação e as atribuições das tarefas. Além disso, os pares possuindo um nível de habilidade semelhante faz com que ambos se sintam mais confortáveis, permitindo obter resultados positivos quanto a motivação e a participação [Tsompanoudi et al. 2019]. O trabalho de Tsompanoudi et al. apresenta dados que demonstram que o nível de experiência é um fator de impacto ao se analisar a programação em pares, de modo que ao realizar um pareamento, a experiência dos desenvolvedores também deve ser levada em consideração.

O estudo de Walter et al. (2015) realizou uma análise em uma empresa de desenvolvimento de software a fim de identificar estratégias para melhorias da gestão de equipes de desenvolvimento. Com base nas análises realizadas, o estudo constatou que

o WIP é um fator de impacto e que deve ser analisado quando se está trabalhando com estratégias de desenvolvimento [Walter et al. 2015]. Dessa forma, como este trabalho realiza uma análise sobre uma prática de desenvolvimento, o WIP deve ser levado em consideração.

Em Stadler et al. (2019), foi realizado um estudo sobre os desafios, benefícios e recomendações para a programação em pares distribuída. Na pesquisa, foi constatado que a utilização de ferramentas é fundamental para a interação entre os membros dos pares, pois elas funcionam como disseminadores de informações permitindo manter os membros mais próximos devido a distância física. Além disso, é ressaltada a importância de estratégias de comunicação alternativas para caso as ferramentas escolhidas inicialmente enfrentam problemas conexão, instabilidade ou queda de serviço [Stadler et al. 2019]. Esse estudo fornece informações importantes para a escolha de ferramentas para a aplicação da programação em pares distribuída como as formas de comunicação por voz, vídeo, chat, tela e compartilhamento de arquivos, uma vez que a ferramenta é o principal meio de comunicação entre os membros do par.

O estudo de Saltz et al. (2020) aborda características necessárias sobre a ferramenta utilizada para o desenvolvimento de programação em pares distribuídas, além de fatores humanos que podem afetar o experimento. Levando em consideração os fatores humanos apresentados, que servem como base para as avaliações deste estudo, destaca-se o fato de que os roteiros de desenvolvimento muito restritivos aplicados pelo pesquisador podem diminuir a motivação dos participantes interferindo na criatividade, pensamento lúdico e exploratório. Além disso, pode ocorrer a demora ou a não troca de funções entre o piloto e copiloto quando há uma pessoa dominante ou quando o tempo necessário para fazer *upload* e *download* do código para o novo piloto é grande [Saltz and Heckman 2020].

4. Materiais e Métodos

O estudo conduzido neste trabalho é uma análise experimental, pois busca compreender quais fatores determinam a aplicação efetiva de programação em pares. Os fatores analisados são: a complexidade das tarefas, número de itens em WIP e a experiência de profissionais. Dessa forma, ao analisar os resultados gerados pela variação dos níveis desses fatores, é possível verificar quais os efeitos deles sobre a programação em pares e, assim, identificar os cenários adequados para a sua aplicação. O estudo é conduzido no contexto de programação em pares realizada de forma remota, em que os programadores não estão no mesmo espaço físico. Nesse sentido, a pesquisa realizada também é considerada exploratória para o ambiente usado.

4.1. Ferramentas para Programação em Pares de Forma Remota

Para este estudo, a ferramenta utilizada como ambiente de colaboração é o Microsoft Teams. A razão da escolha dessa ferramenta é o conjunto de suas funcionalidades que permite a utilização dos recursos necessários para a programação em pares. Tais recursos são: compartilhamento de áudio, vídeo, tela, arquivos, texto, acesso remoto, gravação das conferências e opção para levantar a mão durante as chamadas. Tais recursos ajudam na coordenação da interação e comunicação durante a atividade de programação. Dessa forma, o estudo com os participantes é conduzido dentro de equipes na ferramenta Microsoft Teams.

As tarefas solicitadas aos programadores durante o estudo são implementadas na linguagem C#. A escolha da linguagem C# se deve pelo paradigma orientado a objetos que é bastante difundido na área da programação e no mercado de desenvolvimento de software [Cass 2019]. A codificação no experimento é realizada no Ambiente de Desenvolvimento Integrado (IDE, do inglês *Integrated Development Environment*) Visual Studio 2019, que é uma ferramenta recomendada pela Microsoft para o desenvolvimento em C#. Essa ferramenta também possui a funcionalidade de análise de código e o compartilhamento de código em tempo real com o Live Share.

4.2. Procedimentos

Para a realização do estudo, é elaborado um questionário online a fim de identificar e selecionar os participantes para o experimento. As questões apresentadas no questionário se encontram no Apêndice A. O questionário foi implementado usando a ferramenta Google Forms[1]. A seleção dos participantes é feita levando em consideração perguntas sobre sua área de atuação, tempo de exercício da profissão, se já trabalhou com a linguagem C#, e qual seu nível de experiência com essa linguagem. Nessa linha, o mesmo foi questionado em relação à sua experiência com as ferramentas Microsoft Teams e Visual Studio 2019.

O questionário de convite para a participação no estudo foi enviado a 43 pessoas com experiências diferentes, empregando uma amostragem não probabilística de bola de neve, por meio da qual os participantes selecionados indicam outros participantes. O intervalo de tempo de compartilhamento do questionário foi entre 22/09 a 04/10 de 2020. Do total de pessoas convidadas neste intervalo de tempo, 23 aceitaram participar do estudo. Este número de participantes permite formar 9 pares e dois profissionais com nível de experiência maior e outros dois com nível de experiência menor atuando sozinhos. Assim sendo, é possível validar o desempenho dos pares em relação ao desenvolvimento individual. Com base nos resultados do questionário, os pares são divididos da seguinte forma:

- 3 pares de desenvolvedores experientes.
- 3 pares de desenvolvedores inexperientes.
- 3 pares de desenvolvedores mistos, ou seja, 1 desenvolvedor experiente com outro inexperiente.
- 2 desenvolvedores experientes atuando individualmente.
- 2 desenvolvedores inexperientes atuando individualmente.

A formação dos pares foi feita pelo pesquisador. Essa formação considera o nível de experiência dos profissionais com C#. Além disso, alguns pares já se conheciam devido a forma do compartilhamento do questionário. Entretanto, a busca pelos profissionais considerou diferentes empresas. A Tabela 1 apresenta a distribuição dos desenvolvedores de acordo com a experiência em C#.

Após a seleção dos participantes, é feito o experimento que ocorreu entre 03/10 a 16/10 de 2020. Cada participante selecionado recebe um tutorial onde é ensinado a realizar a instalação do Visual Studio 2019, a criação de conta e ingresso na equipe da plataforma Microsoft Teams. A instalação do Visual Studio 2019 é feita previamente

[1]Questionário para seleção de participantes, disponível na URL `https://docs.google.com/forms/d/1GdY_4mXIYweBv0eErml5X-0qvENvU9gT6G1HWhp-_fw/edit`

Tabela 1. Distribuição da Experiência dos Desenvolvedores em C# em uma Escala de 1 (Pouca Experiência) a 5 (Muita Experiência).

	Par Experiente		Par Inexperiente		Par Misto		Ind. Experiente	Ind. Inexperiente
	Dev. 1	Dev. 2	Dev. 1	Dev. 2	Dev. 1	Dev. 2	Dev. 1	Dev. 1
Experiência	4	4	2	2	5	1	3	1
	5	3	2	1	4	2	4	1
	5	5	1	1	4	1	-	-

pelo participante. Há um tutorial para programadores que participam do estudo em pares[2] e outro para os programadores que participam do estudo de forma individual[3].

Na realização do experimento, cada dupla ou participante individual recebe 4 tarefas de desenvolvimento de *software* para serem implementadas. Cada tarefa é uma especificação[4] em forma de estórias de usuários e requisitos não funcionais. Considerando essas tarefas, a complexidade delas e o WIP são variados de modo a cobrir as seguintes possibilidades:

1. Tarefa de complexidade baixa com WIP definido em 1;
2. Tarefa de complexidade baixa com o WIP definido em 3;
3. Tarefa de complexidade alta com o WIP definido em 1;
4. Tarefa de complexidade alta com o WIP definido em 3;

A tarefa de complexidade baixa com WIP 1 é o desenvolvimento de uma calculadora para realizar operações matemáticas básicas. Já, a outra tarefa de complexidade baixa com WIP 3 é o desenvolvimento de um sistema para realizar conversões de unidades de medidas de comprimento. Para o desenvolvimento da tarefa de alta complexidade com WIP 1, deve-se desenvolver uma aplicação Web para cadastrar e monitorar filmes. Por fim, a tarefa de complexidade alta com o WIP definido em 3 é a implementação em Windows Forms do Jogo da Velha. A definição do WIP em 1 é o próprio desenvolvimento de uma estória da especificação. Já o WIP definido em 3 é composto pelo desenvolvimento de 3 requisitos funcionais da especificação, simultaneamente.

Dentro da equipe no Microsoft Teams, os participantes encontram um arquivo com as 4 tarefas. No arquivo, a ordem das tarefas é aleatória. A aleatorização visa mitigar efeitos que um eventual cansaço dos participantes ao longo do experimento possa ter na tempo de execução da tarefa. Além disso, durante o experimento, o responsável pela pesquisa avaliava as chamadas realizadas e as postagens de arquivos pelos participantes a fim de averiguar a troca de papéis entre os desenvolvedores.

Após o desenvolvimento de cada tarefa, cada participante individualmente ou par de participantes realiza a publicação, no Microsoft Teams, da versão final do código produzido em um arquivo compactado. Após a conclusão das 4 tarefas, é feita a verificação se

[2]Tutorial para os programadores em pares, disponível na URL `https://docs.google.com/presentation/d/1Q-55imguUWbkGNdJ9Q3bKWyGwkhOEhSuD5sAxasWPPc/edit#slide=id.p`

[3]Tutorial para os programadores individuais, disponível na URL `https://docs.google.com/presentation/d/1OGOTsxxl0dDZzl-dG9_y5FuZ2qsHlC3dDfVyitnjYUY/edit#slide=id.p`

[4]Tarefas para Desenvolvimento, disponível na URL `https://drive.google.com/file/d/1gqFX_0DRtxjjmQHiLn8_nEnB1OQzeug8/view?usp=sharing`

os participantes seguiram os protocolos estabelecidos, caso contrário os artefatos produzidos são descartados do estudo. Para os participantes que seguiram as instruções corretamente, o código desenvolvido é classificado em três categorias: Concluído, Incompleto (Falta do Requisito X) e Desistência. O código é classificado como Concluído quando todos os requisitos foram desenvolvidos corretamente. O código é classificado como Incompleto quando ocorre a falta de implementação de um determinado requisito. Por fim, há Desistência quando o desenvolvedor não implementa a tarefa.

Esses materiais e procedimentos envolvem a participação de seres humanos. Dessa forma, eles foram submetidos ao Comitê de Ética em Pesquisa da PUC Minas e aprovado com o número CAAE 34762620.8.0000.5137.

4.3. Métricas e Avaliação

Quatro métricas de avaliação da prática de programação em pares são utilizadas neste estudo, são elas: percepção de impacto do WIP, tempo médio de desenvolvimento, complexidade do código gerado e índice de manutenibilidade do código. A seguir as métricas são explicadas:

- **Impacto do WIP**. Para calcular a percepção dos desenvolvedores em relação ao impacto do WIP, cada participante responde à seguinte questão após a conclusão de cada tarefa: *No contexto de desenvolvimento da especificação, como você avalia o impacto do WIP? Em uma escala de 1 a 5, onde quando mais próximo de 1 menor foi o impacto e quanto mais próximo de 5 maior foi o impacto.*
- **Tempo de desenvolvimento**. Ele é calculado com base no registro do início da implementação pelos participantes registrada na plataforma e a diferença entre os tempos da entrega da implementação de cada tarefa, que é postada no Microsoft Teams.
- **Manutenibilidade do código**. Ela é calculada apelo Visual Studio 2019. Ela possui um valor entre 0 e 100, na qual quanto mais próximo de 100 maior é a facilidade de se realizar alterações. Uma classificação do código no intervalo de 20 a 100 indica um código fácil de se modificar, entre 10 e 19 indica um nível mediano. Já um valor entre 0 e 9 demonstra que é difícil de serem realizadas manutenções.
- **Complexidade de código**. Ela também é calculada pelo Visual Studio 2019. Ela é definida pela complexidade ciclomática, que é o número de caminhos linearmente independentes da execução de um programa [McCabe 1976]. Quanto menor o valor gerado, menor é a complexidade.

5. Resultados

Nesta seção são apresentados os resultados do estudo. Primeiro, analisa-se as características dos participantes do estudo. Após isso, os resultados dos efeitos do WIP, complexidade e experiência no desempenho da programação em pares são apresentados.

5.1. Características dos participantes

A Figura 1 mostra a distribuição dos participantes por área de atuação. Conforme essa figura mostra, a maior proporção dos participantes atuam como Programadores *Frontend* e *Backend* (52,2%). No que se refere ao tempo de profissão, a maior proporção dos participantes exerce a profissão há 2 anos (47,82%, conforme a Figura 2).

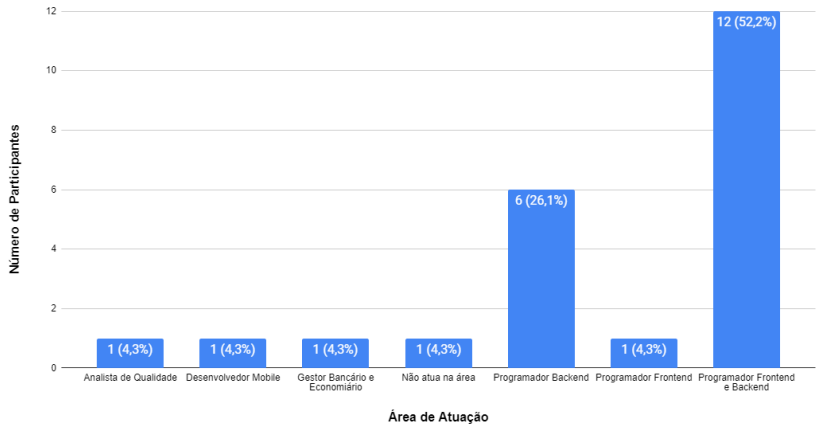

Figura 1. Distribuição dos participantes por áreas de atuação

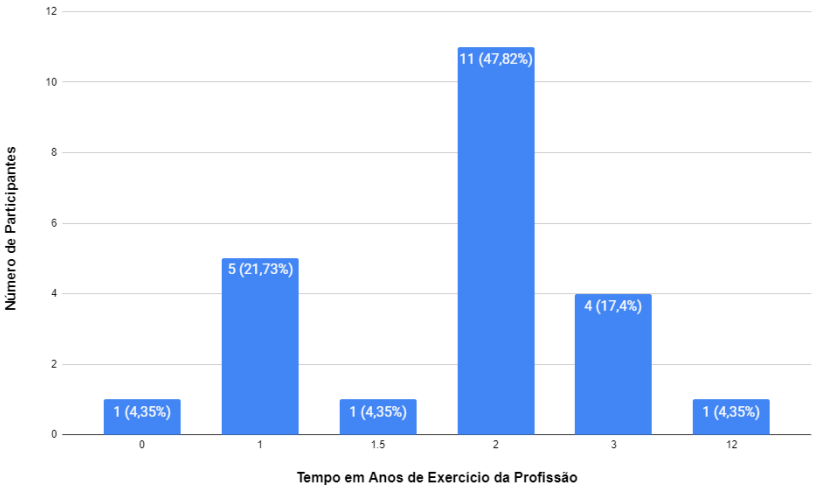

Figura 2. Distribuição dos participantes por tempo de exercício da profissão.

Também analisamos a experiência com a linguagem C# e com as ferramentas Microsoft Teams e Visual Studio. Em uma escala de experiência que vai 1 a 5, sendo 5 muita experiência, maior parte dos participantes apontam ter pouca experiência com C# (resposta 2, 18,18% dos participantes, e 1, 31,81% dos participantes)), muita experiência com o Microsoft Teams (resposta 4, 43,5% dos participantes, e 5, 21,27% dos participantes) e com a IDE Visual Studio (resposta 4, 39,1% dos participantes, e 5, 12% dos participantes).

5.2. Efeito da Complexidade da Tarefa

Analisa-se agora o efeito da complexidade da tarefa nas métricas de tempo de desenvolvimento e índice de manutenibilidade dos grupos em relação às tarefas de baixa e alta complexidade. Desenvolvedores individuais e em pares atuaram em tarefas que possuem diferentes complexidades.

A Figura 3 apresenta em seu eixo X os grupos de desenvolvedores e a complexidade das tarefas, enquanto no eixo Y é representado a média do tempo de desenvolvimento em minutos. Nessa figura, observa-se uma tendência que programadores em pares

gastam em média mais tempo que programadores individuais para desenvolver as tarefas, independente se são de alta ou baixa complexidade. Esse fator é justificado pelo tempo de sincronização entre os programadores que atuam em pares, esse tempo tende a ser alto dado o primeiro contato de programação entre esses pares e a baixa experiência deles com a prática de programação em pares. Apesar dos valores da médias serem distantes, em razão do tamanho da amostra e da alta variação do tempo, a diferença não é significativa para um nível de confiança de 95%.

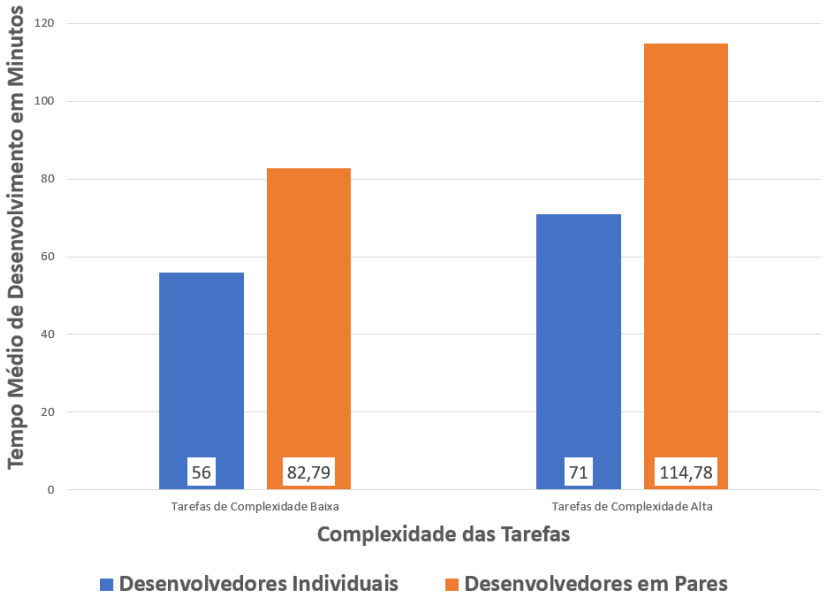

Figura 3. Média do tempo de desenvolvimento das tarefas por desenvolvedores individuais, desenvolvedores em pares e complexidade da tarefa.

Na Figura 4, por sua vez, o eixo X apresenta os grupos de desenvolvedores e a complexidade das tarefas, enquanto no eixo Y apresenta a média do índice de manutenibilidade e os os respectivos intervalo de confiança. Para tarefas de complexidade baixa, há a tendência de que o índice de manutenibilidade seja menor. Para tarefas de complexidade alta, há maior variação e, na média, os índices de manutenibilidade são equivalentes.

5.3. Efeito das Características do WIP

Os dados obtidos a partir da avaliação feita pelos programadores acerca do impacto do WIP durante o processo de desenvolvimento foram agrupados com base nos desenvolvedores individuais e em pares. O objetivo desse agrupamento é a comparação do impacto que o WIP tem em cada um destes cenários por meio do cálculo da média para cada grupo conforme representado na Figura 5. O eixo X apresenta os grupos de desenvolvedores, enquanto o eixo Y apresenta a média dos valores em escala likert de impacto. Pode-se observar uma tendência média de que os programadores que atuaram de forma individual sentiram maior impacto da WIP sob a programação tarefas. No entanto, esse resultado não tem significância estatística quando se considera a margem de erro apresentada para o intervalo de confiança de 95%.

A Figura Figura 6 mostra o número do WIP no eixo X e a média da percepção de impacto do WIP no eixo Y. Com base na figura, é possível observar que quando se

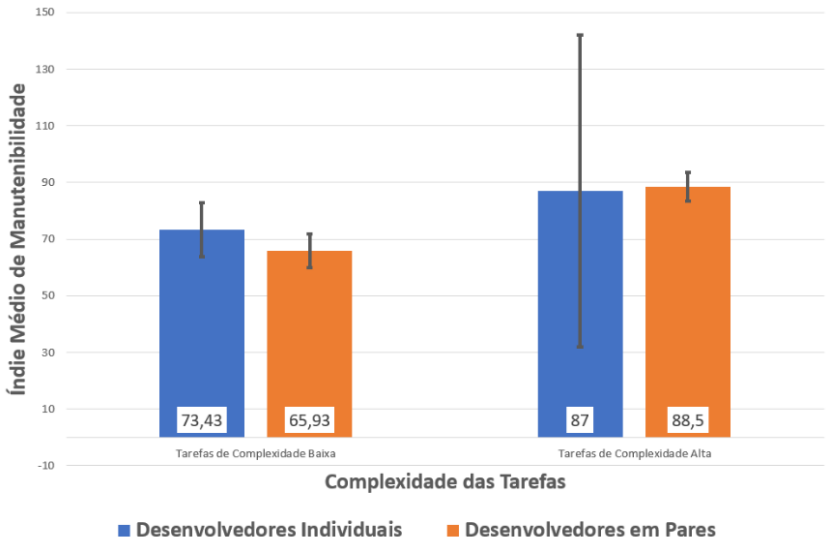

Figura 4. Média do índice de manutenibilidade das tarefas por desenvolvedores individuais e desenvolvedores em pares.

trabalha com um número de WIP mais elevado os programadores tendem a sentirem mais o impacto durante o desenvolvimento.

5.4. Efeito da Experiência de Desenvolvedores de Software

Também é importante analisar o efeito da experiência dos desenvolvedores na programação em pares e os efeitos dela no tempo de desenvolvimento, no índice manutenibilidade e complexidade de código produzido. A Tabela 2 apresenta os resultados obtidos destacando os valores máximo, mínimo e médio de cada uma das métricas. Em pares mistos, aqueles compostos por um programador experiente e um programador inexperiente, observou-se o maior valor médio de complexidade de código e de tempo de desenvolvimento. A formação de pares com essa característica heterogênea pode afetar negativamente essas duas métricas.

Tabela 2. Índice de manutenibilidade, índice de complexidade e tempo de desenvolvimento em diferentes composições de programação individual e em pares.

Desenvolvedores	Manutenibilidade			Complexidade			Tempo		
	Média	Mín.	Máx.	Média	Mín.	Máx.	Média	Mín.	Máx.
Ind. Inexperiente	68,75	62	76	25,00	10	43	66,50	20	94
Ind. Experiente	82,60	67	90	53,80	32	115	53,60	26	78
Par Misto	70,20	48	82	92,60	21	234	101,40	50	180
Par Inexperiente	71,75	62	93	45,00	10	110	93,25	50	210
Par Experiente	74,25	58	89	38,00	15	126	90,88	20	210

Também é possível verificar que, em média, os desenvolvedores experientes produzem códigos com um menor tempo médio de desenvolvimento (53,6) e maior índice médio de manutenibilidade (82,6). Entretanto, os desenvolvedores inexperientes geraram códigos menos complexos em relação aos experientes. Uma das razões para esse resultado é que os desenvolvedores experientes tendem a utilizarem técnicas de programação mais complexas e avançadas, fazendo com que tarefas simples se tornem mais complexas.

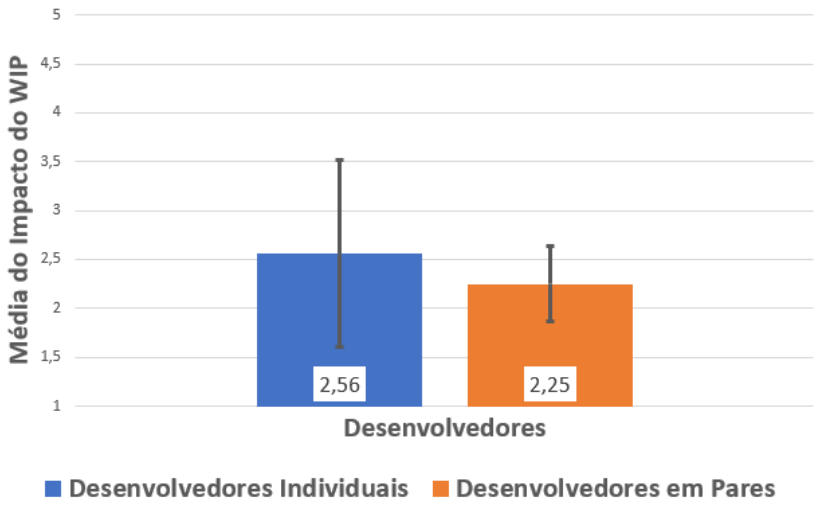

Figura 5. Distribuição da Média da Percepção de Impacto do WIP por Desenvolvedores Individuais e Desenvolvedores em Pares

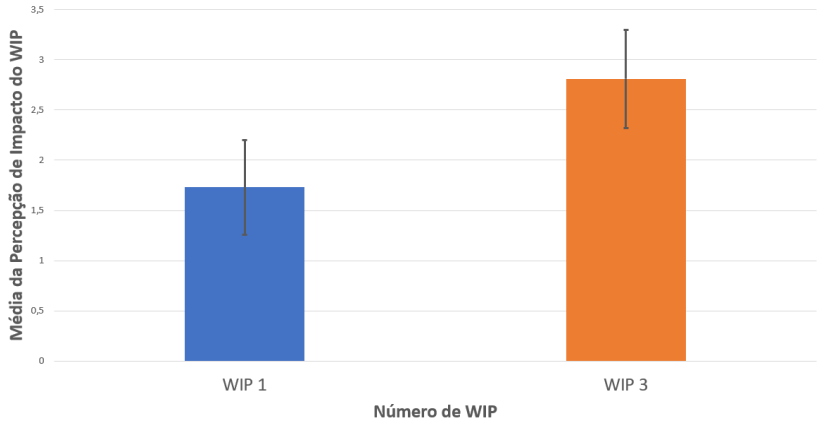

Figura 6. Distribuição da Média da Percepção do Impacto do WIP pelo Número de WIP

6. Discussão

Está se tornando comum que equipes se adaptarem para trabalharem de forma remota. Entender como a prática de programação em pares se insere nesse contexto é relevante. Este estudo analisa cenários do emprego dessa prática e apresenta *insights* relevantes nesse contexto. O estudo realizado faz uma análise da experiência de desenvolvedores de *software*, WIP e complexidade de tarefas na programação em pares de forma exploratória. Os resultados obtidos mostram os cenários de uso e desempenho dessa prática.

Os resultados obtidos indicam que a programação em pares pode ser adequada em cenários que o valor do WIP tem grande impacto no desenvolvimento e se admita um tempo de desenvolvimento maior. A formação dos pares deve levar em consideração os objetivos desejados, por exemplo, caso o objetivo seja a produção de um *software* em menos tempo e com índice de manutenibilidade mais elevado um par experiente é o mais adequado. A partir da análise do impacto do WIP percebido pelos desenvolvedores individuais em relação aos que atuaram em pares, é possível observar que esse impacto durante o processo de desenvolvimento é menor quando se está utilizando a prática da

programação em pares. A redução do impacto percebido pelos programadores que seguiram a programação em pares mostra que se torna mais fácil o gerenciamento das tarefas em desenvolvimento quando se usa essa prática. Além disso, ao se trabalhar com um valor menor de WIP, a percepção do impacto durante o desenvolvimento pelos desenvolvedores é menor. Dessa forma, um valor menor de WIP torna o processo de desenvolvimento mais organizado.

Os efeitos da complexidade das tarefas sobre o desenvolvimento em pares em relação ao desenvolvimento individual mostram que o tempo de desenvolvimento dos programadores em pares é relativamente maior que os individuais. Dentre os fatores que justificam o aumento no tempo de desenvolvimento, destaca-se o tempo gasto com o compartilhamento de conhecimento entre cada integrante da dupla. Ao se tratar do índice de manutenibilidade, destaca-se a diferença das tarefas de complexidade baixa dos desenvolvedores individuais em relação aos pares.

Durante o experimento, foi avaliado se todos os participantes seguiram os protocolos estabelecidos. Um participante não seguiu os protocolos corretamente durante o desenvolvimento de uma das tarefas e com isso, a sua especificação foi desconsiderada e removida da análise. O que se observa em relação a isso é que alguns participantes perdem facilmente o foco da programação em pares tentando ganhar contexto de outras práticas corriqueiras mas que não são objetos de avaliação do estudo, como é o caso do desenvolvimento dirigido por testes (TDD, do inglês *Test-Driven Development*). A alta variação observada nos resultados são indicativos de que os resultados do emprego dessa prática pode ser sensível à outras variáveis, como a experiência que um programador tem de trabalhar com o outro e a experiência que eles têm com a programação com pares

Por fim, enquanto limitação, ressalta-se que o estudo conduzido tem um caráter experimental. A análise do impacto das ferramentas utilizadas não foi realizado, pois este não é o objetivo deste trabalho. Ele trata de cenários de uso da programação em pares, mas sem um detalhamento com alta amostragem em nenhum deles. Enquanto o estudo contribui com *insights* relevantes, ele não permite tirar conclusões definitivas sobre os cenários avaliados. Além disso, análises qualitativas como os efeitos de fatores humanos como confiança, respeito, experiências prévias entre os participantes, conhecimento comum do código e a troca de funções entre os participantes não foram avaliados. Esses são aspectos fundamentais de serem abordados em novos trabalhos nesse campo de estudo.

7. Conclusão e Trabalhos Futuros

Este trabalho buscou avaliar cenários de uso da programação em pares levando em consideração o WIP, experiência do desenvolvedor e a complexidade da tarefa para identificar o cenário mais adequado para a aplicação dessa prática. Realizou-se um experimento com o objetivo de avaliar cenários da aplicação da programação em pares e comparar os resultados com o desenvolvimento solo. Considerou-se as métricas tempo de desenvolvimento, no índice de manutenibilidade, no índice de complexidade e na percepção que o desenvolvedor tem quanto ao impacto do WIP. Os resultados apontam que o cenário de uso de programação em pares tem efeito sobre o complexidade do código gerado, tempo de desenvolvimento e índices de manutenibilidade do código.

Os resultados obtidos permitiram explorar cenários e levantar novas questões sobre a programação em pares. Em relação aos trabalhos futuros, pode-se buscar uma abor-

dagem quantitativa que amplie amostragem em cada cenário e uma abordagem qualitativa que dê perspectivas sobre fatores humanos relevantes em cada cenário. No contexto dos cenários estudos, uma comparação sobre a abordagem de programação em pares remota e abordagem de programação em pares não remota também se mostra relevante. Os dados obtidos nesta pesquisa estão disponíveis[5] para que seja possível utilizá-los em eventuais evoluções deste estudo.

Referências

Ajami, S., Woodbridge, Y., and Feitelson, D. G. (2017). Syntax, predicates, idioms - what really affects code complexity? In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 66–76.

Alshayeb, M. (2009). Empirical investigation of refactoring effect on software quality. *Information and software technology*, 51(9):1319–1326.

Anwer, F., Aftab, S., Shah, S. S. M., and Waheed, U. (2017). Comparative analysis of two popular agile process models: Extreme programming and scrum. *International Journal of Computer Science and Telecommunications*, 8:1–7.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition.

Cass, S. (2019). The top programming languages 2019. *Spectrum IEEE*.

Chaouch, S., Mejri, A., and Ghannouchi, S. (2019). A framework for risk management in scrum development process. *Procedia Computer Science*, 164:187–192.

Coman, I. D., Robillard, P. N., Sillitti, A., and Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software*, 91:124–134.

Hannay, J. E., Dyba, T., Arisholm, E., and Sjøberg, D. I. (2009). The effectiveness of pair programming: A meta-analysis. *Information and software technology*, 51(7):1110–1122.

IEEE (1993). Ieee standard for a software quality metrics methodology. *IEEE Std 1061-1992*, pages 1–96.

ISO/IEC (2010). Iso/iec 25010 system and software quality models. Technical report.

Kearney, J. K., Sedlmeyer, R. L., Thompson, W. B., Gray, M. A., and Adler, M. A. (1993). Software complexity measurement. *Communications of the ACM*, 36:81–95.

Man, L. K. and Chan Keith, C. (2006). Software process fusion: Uniting pair programming and solo programming processes. *Proceedings of SPW/ProSim*, pages 115–123.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., and Balik, S. (2003). Improving the cs1 experience with pair programming. *ACM SIGCSE Bulletin*, 35(1):359–362.

[5]Dados obtidos na pesquisa, disponível na URL <https://drive.google.com/drive/folders/1OEc-51aU_PsDGn5IWie1BMa-6bQj_FX8?usp=sharing>

Nawrocki, J. and Wojciechowski, A. (2001). Experimental evaluation of pair programming. *European Software Control and Metrics (Escom)*, pages 99–101.

Newkirk, J. (2002). Introduction to agile processes and extreme programming. *International Conference on Software Engineering (ICSE)*, pages 695–696.

Pressman, R. S. (2011). *Engenharia de Software: Uma Abordagem Profissional*. Amgh, 7th edition.

Qureshi, M. R. J. and Ikram, J. S. (2015). Proposal of enhanced extreme programming model. *International Journal of Information Engineering and Electronic Business*, 7(1):37.

Saltz, J. and Heckman, R. (2020). Using structured pair activities in a distributed online breakout room. *Online Learning*, 24(1):227–244.

Schümmer, T. and Lukosch, S. G. (2009). Understanding tools and practices for distributed pair programming. *Journal of Universal Computer Science, 15 (16), 2009*.

Sereno, B., da Silva, D. S. A., Leonard, D. G., and Sampaio, M. (2011). Método híbrido conwip/kanban um estudo de caso. *Gestão e Produção*, 18(3):651–672.

Stadler, M., Vallon, R., Pazderka, M., and Grechenig, T. (2019). Agile distributed software development in nine central european teams: Challenges, benefits, and recommendations. *International Journal of Computer Science & Information Technology (IJCSIT) Vol*, 11.

Tsompanoudi, D., Satratzemi, M., Xinogalos, S., and Karamitopoulos, L. (2019). An empirical study on factors related to distributed pair programming. *International Journal of Engineering Pedagogy*, 9.

Walter, M., Tramontini, R., Fontana, R. M., Reinehr, S., and Malucelli, A. (2015). From sprints to lean flow: Management strategies for agile improvement. In *International Conference on Agile Software Development*, pages 310–318. Springer.

Wang, X., Conboy, K., and Cawley, O. (2012). "leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, 85(6):1287–1299.

Xinogalos, S., Satratzemi, M., Chatzigeorgiou, A., and Tsompanoudi, D. (2019). Factors affecting students' performance in distributed pair programming. *Journal of Educational Computing Research*, 57(2):513–544.

APÊNDICE A - Questionário Aplicado Para Seleção de Candidatos para o Experimento

1. Qual sua área de atuação?
   (a) Arquiteto de *Software*
   (b) Programador *Backend*
   (c) Programador *Frontend*
   (d) Programador *Frontend e Backend*
   (e) Outros
2. Quantos anos você exerce essa profissão?
3. Você já trabalhou com a linguagem C#?

4. De 1 a 5, qual o seu nível de experiência com C#?
    (a) 1 (Pouca Experiência)
    (b) 2
    (c) 3
    (d) 4
    (e) 5 (Muita Experiência)
5. Você já possui cadastro na plataforma Microsoft Teams?
    (a) Sim
    (b) Não
6. De 1 a 5, qual o seu nível de experiência com o Microsoft Teams?
    (a) 1 (Pouca Experiência)
    (b) 2
    (c) 3
    (d) 4
    (e) 5 (Muita Experiência)
7. Você possui o Visual Studio 2019 instalado no computador?
    (a) Sim
    (b) Não
8. De 1 a 5, qual o seu nível de experiência com o Visual Studio 2019?
    (a) 1 (Pouca Experiência)
    (b) 2
    (c) 3
    (d) 4
    (e) 5 (Muita Experiência)
9. Deixe seu contato.