Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Herez Moise Kattan

Programação e revisão simultânea em Pares: uma extensão à programação em par

São Paulo
2015

Herez Moise Kattan

Programação e revisão simultânea em Pares: uma extensão à programação em par

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Computação.

Área de Concentração: Engenharia de *Software*.

Data da aprovação _____/_____/_____

_____
Prof. Dr. José Eduardo Zindel Deboni (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. José Eduardo Zindel Deboni (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dr. Alfredo Goldman vel Lejbman (Membro)
IME-USP – Instituto de Matemática e Estatística da Universidade de São Paulo

Prof. Dr. Eduardo Martins Guerra (Membro)
INPE – Instituto Nacional de Pesquisas Espaciais

Herez Moise Kattan

Programação e revisão simultânea em Pares: uma extensão à programação em par

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Computação.

Área de Concentração: Engenharia de *Software*.

Orientador: Prof. Dr. José Eduardo Zindel Deboni

São Paulo
Dezembro/2015

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

K19p **Kattan, Herez Moise**
Programação e revisão simultânea em Pares: uma extensão à programação em par. / Herez Moise Kattan. São Paulo, 2015.
88p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software

Orientador: Prof. Dr. José Eduardo Zindel Deboni

1. Programação em pares 2. Engenharia simultânea 3. Teste de software 4. Qualidade do produto 5. Produto de software 6. Tese I. Deboni, José Eduardo Zindel, orient. II. IPT. Coordenadoria de Ensino Tecnológico III. Título

16-14 CDU 004.42(043)

RESUMO

A programação em pares é uma prática para equipes de desenvolvimento de *software* que melhora a qualidade do código desenvolvido, aumenta o nível técnico da equipe, estimula a disseminação do conhecimento, promove a ideia de propriedade e responsabilidade coletiva para o sistema, reduz o número de defeitos do sistema, estimula a satisfação, motivação, colaboração, comunicação e a confiança entre os membros da equipe. Porém, com perda de produtividade em alguns casos. Este trabalho usa a Engenharia Simultânea, *Code Review, Mob Programming e Coding Dojo*, visando aumentar a produtividade no desenvolvimento da programação em pares, criando uma nova técnica vantajosa para programadores de diferentes graus de experiência e tarefas de diversos níveis de complexidade. O trabalho se inicia com a revisão do estado da arte da programação em pares e em grupo, revisão de código e engenharia simultânea. A técnica proposta é baseada nos pontos fortes encontrados na revisão bibliográfica sobre estas técnicas correlatas. São realizados exemplos de aplicação prática, onde a produtividade e qualidade são medidos. Os resultados mostram que a nova técnica é eficaz nos exemplos de aplicação realizados, observando o aumento da produtividade no desenvolvimento e da qualidade do produto de *software*.

Palavras-chave: Técnicas ou práticas de desenvolvimento e teste de *software*; Programação em Pares ou em Grupo; Revisão de Código; Engenharia Simultânea.

ABSTRACT

Programming and review simultaneous in Pairs: a pair programming extension

The pair programming is a practice for software development teams to improve the quality of their code and the technical level of the team, encourages the dissemination of knowledge, it promotes the idea of ownership and collective responsibility for the system, reduces the number system defects, stimulates satisfaction, motivation, collaboration, communication and trust among team members. But its burden is the loss of productivity in some cases. This work uses the Concurrent Engineering, Code Review, Mob Programming and Coding Dojo, aiming to increase productivity in the development of pair programming, creating a new advantageous technique for programmers with different expertise levels and tasks of various levels of complexity. The work begins with the state of the art review of pair/mob programming, code review and concurrent engineering. The propose technique is based on the strengths found in the literature review of this correlative techniques. Examples application are carried out in practical, where productivity and quality are measured. The results show that the new technique is effective in application examples performed by observing the increase in development productivity and quality of the software product.

Key Words: Techniques or practices to software development and software testing; Pair Programming; Mob Programming; Code Review; Concurrent Engineering.

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A programação em pares é uma prática para equipes de desenvolvimento de *software*, é também um processo de revisão informal, melhora a qualidade do código desenvolvido e o nível técnico da equipe. Ela estimula a refatoração, promove a propriedade e responsabilidade coletiva para o sistema e reflete a ideia de Weinberg (1971) de programação sem ego, onde o *software* é de propriedade da equipe como um todo e os indivíduos não são responsáveis pelos problemas com o código; toda a equipe tem responsabilidade em resolver estes problemas.

O código produzido é escrito por pares de programadores, que possuem papéis distintos, sentados lado a lado e usando o mesmo computador. Um parceiro é responsável pela codificação e pensa nos algoritmos e na lógica de programação. O outro parceiro observa o código produzido e tenta pensar mais estrategicamente em como melhorá-lo e torná-lo mais simples, além de verificar possíveis erros e pontos de falha. (BECK, 1999)

A ideia é desenvolver software em duplas de programadores e não como uma atividade isolada de uma só pessoa. Dois desenvolvedores dividem o mesmo teclado e o mesmo monitor. Com isso promove-se aspectos sociais da programação, enquanto um cria o código o outro desenvolvedor pode revisar, discutir alternativas, verificar a qualidade do código e ajudar em passagens complexas. Segundo Cockburn et al. (2000) a programação em pares envolve comunicação e colaboração constantes. Estimula a disseminação do conhecimento entre os membros da equipe.

A programação em pares promove a confiança entre os programadores (SLATEN et al., 2004). Bella et al. (2013) constatou em seu trabalho redução do número de defeitos ao usar a programação em pares. Segundo Plaue et al. (2013) a programação em pares estimula o companheirismo. É um fator de proteção para conservação e diminuição da evasão de mulheres e programadores novatos nos cursos de ciências da computação. Pode ser uma intervenção de baixo custo para a redução do *turn-over* em equipes de *software*. Jones et al. (2013) observa que o parceiro que não está na posse do teclado, contribui com boas ideias e Sillitti et al. (2012) que ajuda a manter disciplina e concentração em atividades produtivas.

Conforme Dyba et al. (2007) o produto desenvolvido pela programação em pares é, em geral, de qualidade superior a métodos convencionais. Porém,

constatou-se também ao revisar quinze outras pesquisas sobre programação em pares, que ocorre perda de produtividade ao usar a técnica, especialmente para programadores experientes e tarefas simples. Observa-se ao usar a programação em pares, o aumento do esforço no projeto, se comparado à programação individual.

A perda de produtividade é uma crítica frequente à programação em pares, uma vez que se está dedicando dois profissionais para executar o mesmo trabalho. Há uma constante discussão nos meios acadêmicos e profissionais se as vantagens da técnica, justificam esta perda de produtividade.

Existem técnicas correlatas, como o *mob programming,* onde toda a equipe participa em torno de um computador (ZUILL, 2012) e a revisão de código em pares, que atualmente são alternativas analisadas, inclusive acerca da produtividade obtida em relação à programação em pares. (SWAMIDURAI, DENNIS e KANNAN, 2014)

Em 1982 foi iniciado um estudo, conduzido pelo DARPA (*Defense Advanced Research Project Agency* – Agência de Projetos de Investigação Avançada de Defesa), buscando uma forma de aumentar o grau de paralelismo das atividades de desenvolvimento de produtos. O resultado deste trabalho, definiu o termo Engenharia Simultânea como:

> Engenharia Simultânea é uma abordagem sistemática para o desenvolvimento integrado e paralelo do projeto de um produto e os processos relacionados, incluindo manufatura e pós-venda Esta abordagem procura fazer com que as pessoas envolvidas no desenvolvimento considerem, desde o início, todos os custos, prazos e requisitos dos clientes. (WINNER et al., 1988)

A Engenharia Simultânea é baseada na ideia do processamento paralelo/simultâneo dos processos da empresa, que reduz o tempo de lançamento de um novo produto bem como a melhoria da qualidade.

Os mecanismos básicos para a implementação dos processos em paralelo/simultâneo são as equipes multifuncionais (também chamadas de “força-tarefa”), que trabalham simultaneamente juntas, e a correspondente troca de informações tecnológicas e ferramentas para suportar os processos de Engenharia Simultânea. (PITHON, 2004)

Uma possível alternativa para o aumento da produtividade na programação em pares é o uso do paralelismo e das equipes multidisciplinares da Engenharia Simultânea, para que o desenvolvimento seja feito de maneira simultânea. Outra possibilidade é a incorporação de um processo de revisão de código baseado em pares e, uma fase para projetar o emparelhamento do desenvolvimento simultâneo.

1.2 Objetivo

Propor uma técnica para desenvolvimento e testes de sistemas de *software* mais produtiva em relação à programação em pares, por meio da incorporação de aspectos de Engenharia Simultânea, Revisão de Código em Pares e *Mob Programming*. Avaliar a técnica, e se preserva a qualidade do produto quando comparada à programação em pares.

1.3 Contribuições

A principal contribuição é uma revisão e aprimoramento da programação em pares, na forma de uma nova técnica de organização do trabalho dos programadores com base no paralelismo e equipes multidisciplinares da engenharia simultânea, incorporação de um processo de revisão de código baseado em pares e, uma fase para projetar o emparelhamento do desenvolvimento simultâneo e outra para o descanso reflexivo, prevenção e resolução de conflitos.

O trabalho traz como contribuições complementares, uma revisão bibliográfica da programação em pares e em grupo, revisão de código e engenharia simultânea, novas métricas para produtividade das equipes e qualidade do produto *software*. Produzir mais em menos tempo, gera um ganho de tempo, que além da economia de recursos ou redução do *time-to-benefit,* pode significar a redução do *time-to-market* (tempo transcorrido desde a detecção da necessidade, até a introdução de um novo produto no mercado) do produto de *software* desenvolvido, que são contribuições indiretas deste trabalho.

1.4 Método de trabalho

Para cumprir o objetivo, o método de trabalho adotado é: começar pela revisão do estado da arte da programação em pares, programação em grupo, revisão de código e engenharia simultânea. Em seguida, são estabelecidas as métricas para permitir a avaliação da produtividade no desenvolvimento de *software* e qualidade do produto de *software*. Depois é proposta uma técnica baseada nos pontos fortes encontrados na revisão bibliográfica das técnicas correlatas.

Para avaliar a eficácia da nova técnica são verificados nos exemplos de aplicação se a nova técnica foi mais produtiva e a qualidade do produto de *software*.

A aplicação da nova técnica é realizada em ambiente acadêmico e corporativo. São realizados exemplos de aplicação em empresas de tecnologia da informação, desenvolvimento de *software*, instituições financeiras e instituições de ensino que

permitam mensurar, classificar e relatar a experiência com programação dos participantes, produtividade obtida durante o desenvolvimento de *software* no exemplo de aplicação, qualidade e o número de defeitos em comparação à programação em pares.

É feita a análise dos resultados obtidos nos exemplos de aplicação para com base nestes dados, avaliar a eficácia da nova técnica e na busca por explicações.

Na conclusão, encerra-se a pesquisa discorrendo sobre um resumo dos resultados obtidos nos exemplos de aplicação, as contribuições, uma explanação das limitações e as recomendações para trabalhos futuros.

1.5 Organização do trabalho

A seção 2, Revisão Bibliográfica, apresenta o estado da arte da programação em pares, programação em grupo, revisão de código e engenharia simultânea.

A seção 3, Programação e revisão simultânea em Pares, apresenta a nova técnica, principal objeto de pesquisa deste trabalho. A PrsP é especificada e descrita com intuído de permitir a avaliação e comparação com outras técnicas; é descrito como trabalhar em pares usando uma equipe multidisciplinar; é feita uma explanação sobre a responsabilidade e autoridade dos pares para as decisões, o planejamento e emparelhamento do projeto, o paralelismo durante a execução da tarefa, o processo de revisão do código, o descanso produtivo e reflexivo, bem como o processo sugerido para resolução de conflitos.

A seção 4, Exemplos de aplicação da técnica, apresenta o sumário e os detalhes de todos exemplos de aplicação realizados, concentrando-se na produtividade obtida durante o desenvolvimento de *software,* qualidade e o número de defeitos em comparação à programação em pares. São apresentadas as novas métricas de avaliação da produtividade e qualidade do produto de *software* obtido com a PrsP, baseadas nas ferramentas, medidas, indicadores e pesquisas atuais disponíveis, para comparação da nova técnica à programação em pares com precisão superior a linhas de código por desenvolvedor e contagem de defeitos.

A seção 5, Análise dos resultados e validação da técnica, detalha e analisa os resultados obtidos durante a pesquisa, culminando com a avaliação da nova técnica, baseando-se nos resultados para avaliar sua eficácia e na busca por explicações.

A seção 6, Conclusão, limitações e trabalhos futuros, conclui a pesquisa, discorrendo sobre um resumo dos resultados, das contribuições, limitações e sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Esta seção revisa a literatura da programação em pares (*pair programming*), programação em grupo (*mob programming),* revisão de código *(code review)* e engenharia simultânea (*simultaneous engineering* ou *concurrent engineering*).

2.1 Programação em Pares (*Pair Programming*)

A questão da revisão da literatura da programação em pares é: Quais são os pontos fortes e fracos já observados e/ou constatados da programação em pares?

As palavras chaves utilizadas para pesquisa dos artigos abrangem: pair programming, paired programming, double programming, programação em par(es), programação em dupla(s) e pareamento.

O período pesquisado é de 40 anos, com estudos publicados entre os anos 1975 e 2015.

Cada estudo encontrado foi avaliado conforme o seguinte critério de inclusão: todos os trabalhos cujo objeto de pesquisa é programação em pares são incluídos.

Os seguintes tipos de trabalhos foram excluídos: publicações sem um método de pesquisa, publicações que já tenham sido analisadas numa revisão sistemática ou duplicadas e feitas em idioma diferente do português e inglês.

A definição da programação em pares é que dois programadores trabalham colaborativamente na mesma atividade, sentados lado a lado em frente a um único computador. Enquanto uma pessoa está escrevendo o código, por exemplo, a outra observa atentamente o trabalho produzido, buscando defeitos e sugestões de melhoria. A programação em pares estimula a disseminação do conhecimento, reduz a quantidade de defeitos e gera *software* com mais qualidade. (BEGEL; NAGAPPAN, 2008)

Apesar de ter sido conhecida como uma técnica associada ao método eXtreme Programming, a criação da programação em pares é anterior ao método XP. Existem muitos relatos antigos favoráveis à técnica, como o relato de Brooks (1975) ao programar em par na faculdade entre 1953 e 1956, ele e seu par escreveram 1500 linhas de código, que executaram sem nenhum erro na primeira tentativa.

Coplien e Schmidt (1995) com base em resultados do projeto de investigação Pasteur (um grande estudo sociológico/antropológico com cinquenta organizações de desenvolvimento de *software* altamente eficazes) da *Bell Labs Research*,

publicaram um padrão para organização do trabalho dos programadores chamado "desenvolvimento em pares", os autores deste padrão destacaram que pessoas às vezes sentem que apenas podem resolver um problema se tiverem ajuda. Alguns problemas são maiores do que qualquer indivíduo.

A solução para Coplien e Harrison (2005) é tornar os projetos compatíveis com o emparelhamento de trabalhar em conjunto; desta forma podem produzir mais do que a soma dos dois individualmente.

Rotação dos pares, disseminação do conhecimento e redução dos defeitos

Constantine (1995) publicou a redução do número de defeitos ao programar-se em pares.

Nosek (1998) constatou que a programação em pares melhora a qualidade do código e do algoritmo, porém, constatou também o aumento do esforço no desenvolvimento, pois observou que o par gastou mais horas de trabalho do que se um único programador trabalhasse isoladamente.

Uma vantagem da rotação dos pares é espalhar o conhecimento do sistema pela equipe inteira. Como importante efeito colateral temos também o compartilhamento de técnicas e competências entre os membros da equipe (WILLIANS; KESSLER, 2000).

A inclusão de um novo membro na equipe que não conhece o sistema cria o problema do primeiro contato com o código. A programação em pares contribui para resolução deste problema, pois o novo membro da equipe pode trabalhar em par com alguém mais experiente. (WILLIAMS; KESSLER, 2003)

Os pares são constantemente trocados e os papéis também, objetivando permitir a todos os membros da equipe ter conhecimento sobre todas as partes do sistema, resolvendo o problema de um único desenvolvedor deter o conhecimento sobre uma parte do sistema. Disseminando o conhecimento, mais pessoas participam do desenvolvimento de uma funcionalidade e aprendem como o sistema ou uma ferramenta de desenvolvimento funciona. (BECK et al., 2004)

Bella et al. (2013) publicou um estudo de caso que foi realizado para avaliar o efeito da programação em pares na qualidade e eficiência das correções de defeitos. Foi estudado um projeto de desenvolvimento de *software* em uma grande empresa italiana, por um prazo de quatorze meses. Comparado com estudos de casos existentes de programação em pares, o período mais longo torna o estudo mais

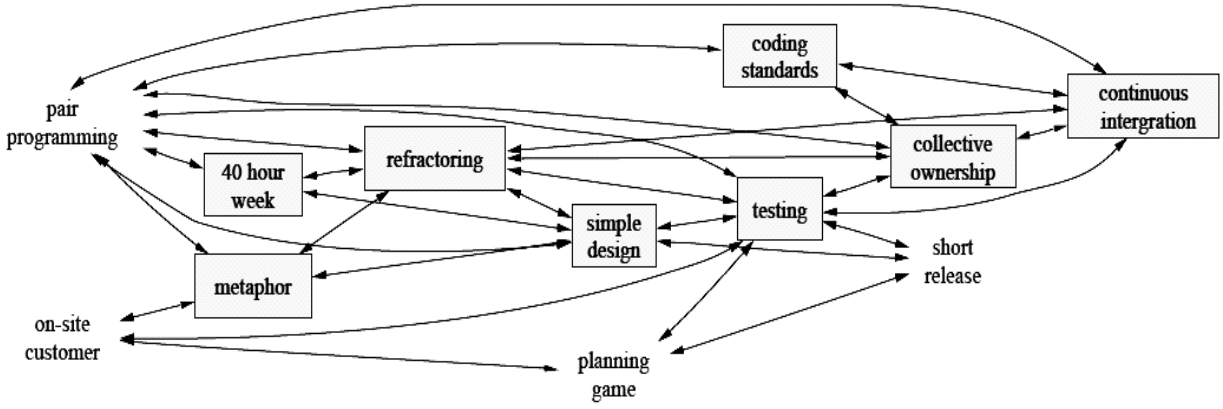
realista. Em análise exploratória, a efetividade da programação em pares foi investigada no contexto de correções de defeitos e implementações de histórias de usuários. A análise mostra que a introdução de novos defeitos tende a diminuir quando a programação em pares é usada. Os resultados são consistentes para ambos os contextos.

Swamidurai e Umphress (2014) constataram aumento da produtividade quando a programação em pares é dinâmica, ou seja, há rotação dos pares na equipe. A produtividade é menor quando a programação em pares é estática, sem rotação dos pares na equipe.

Projeto pequeno, muitos desenvolvedores e período curto de desenvolvimento

A pesquisa conduzida por Smith et al. (2002) reconheceu o importante papel da programação em pares. Ela se relaciona diretamente a quase todas as outras práticas do método XP, conforme ilustra a Figura 1. A centralidade da programação em pares para o método XP é evidenciada devido ao fato de que caso removida afetaria diretamente todas as práticas relacionadas. O trabalho observa o problema de produtividade da programação em pares e conclui que a NASA não deveria usar método ágil baseado em programação em pares em 64% dos seus projetos. Para os demais casos, o método usado na pesquisa apenas endossou vantagem sobre abordagens convencionais em um relativamente pequeno e específico conjunto de casos: quando o projeto é relativamente pequeno, quando existe uma abundância de desenvolvedores e se o período de desenvolvimento for curto.

Figura 1 - Práticas do método XP relacionadas à programação em pares



Fonte: Smith et al. (2002)

Para Padberg e Müller (2003), o código produzido usando a programação em pares é mais livre de defeitos, pois todo o código é revisado por pelo menos um programador. Eles concluem que, quando a pressão do mercado é forte, acrescentar

desenvolvedores e formar mais pares, pode acelerar o projeto e aumentar o seu valor para o negócio, apesar de aumentar também o custo.

Processo de revisão/inspeção informal e comunicação

Para Williams et al. (2000) a utilização da técnica de programação em pares atua como um processo de revisão informal, porque cada linha de código é olhado por pelo menos duas pessoas. Inspeções de código ajudam muito na descoberta de uma elevada percentagem de erros de *software*. No entanto, são demorados para organizar e, geralmente, representam atrasos no processo de desenvolvimento.

Para Cockburn e Williams (2000 e 2001) a programação em pares é um processo de revisão menos formal e, provavelmente, não encontra tantos erros como inspeções de código, porém, é muito mais barato que um processo de inspeção formal. Ela apoia a refatoração, que é um processo de melhoria de *software*. A dificuldade de implementar isso em um ambiente normal de desenvolvimento de *software* é que o esforço em refatoração é gasto no presente, para um benefício a longo prazo no futuro. Quando são utilizados a programação em pares e propriedade coletiva, outros desenvolvedores se beneficiam imediatamente da refatoração e eles tendem a apoiar o processo, pois reconhecem seu benefício.

Cockburn et al. (2001) e Williams et al. (2000) em seus trabalhos usaram programadores iniciantes e observaram que a produtividade, ao usar a programação em pares, parece ser comparável à de duas pessoas trabalhando de forma independente. Uma razão sugerida é que programando em pares discute-se o sistema antes de desenvolvê-lo, então provavelmente ocorrerá menos erros e retrabalho. Além disso, o número de erros evitados pela inspeção informal realizada pela programação em pares é tal que menos tempo é gasto na reparação de *bugs* descobertos durante o processo de testes.

Du et al. (2015) Executaram um experimento de programação básica usando a linguagem de programação C. Os resultados são analisados por meio de entrevistas após a realização do experimento e a conclusão é que a programação em pares é eficaz para melhorar a comunicação em um exercício básico de C.

Experiência do programador e complexidade da tarefa

Arisholm et al. (2007) e Parrish et al. (2004) realizaram seus trabalhos com programadores experientes. Eles descobriram que houve uma significativa perda de produtividade em comparação a dois programadores trabalhando sozinhos. Havia

alguns benefícios na qualidade, mas que não compensavam totalmente a perda da produtividade da programação em pares. No entanto, segundo eles, a partilha de conhecimento que acontece durante a programação em pares é muito importante, pois reduz os riscos globais de um projeto no caso de algum membro da equipe sair e isso pode justificar o seu uso.

Hulkko e Abrahamsson (2005) observaram em quatro estudos de caso a eficácia da programação em pares na disseminação de conhecimento e que ela é mais útil em tarefas complexas.

A pesquisa conduzida por Dyba et al. (2007) revisou quinze trabalhos sobre programação em pares e todos eles usaram tarefas de programação como base para a comparação. O número de indivíduos nos estudos oscilou entre 12 e 295. O estudo concentrou-se nos aspectos: duração, tempo do calendário de desenvolvimento; esforço, mensurado em horas de trabalho; qualidade, quanto melhor ficou o produto final. Ao analisar-se o resultado, constata-se a melhora da qualidade, redução do número de defeitos, porém, observa-se também, ao usar a programação em pares, o aumento do esforço no projeto, se comparado à programação individual.

Conforme é descrito no Quadro 1, há uma relação direta das vantagens da programação em pares com a experiência do programador e complexidade da tarefa, apenas um estudo (apesar de grande e consistente) dos quinze revisados levou em consideração a complexidade da tarefa e experiência do desenvolvedor, há a necessidade de mais estudos que levem-nos em consideração.

Quadro 1 – Diretriz para quando usar a programação em pares

Experiência do programador	Complexidade da tarefa	É vantajoso usar a programação em pares?
Júnior	Simples	Sim, se o aumento da qualidade é o principal objetivo.
Júnior	Complexa	Sim, se o aumento da qualidade é o principal objetivo.
Pleno	Simples	Não.
Pleno	Complexa	Sim, se o aumento da qualidade é o principal objetivo.
Sênior	Simples	Não.
Sênior	Complexa	Não, a menos que a tarefa é complexa demais para ser resolvida por apenas um programador sênior.

Fonte: Dyba et al. (2007)
Legenda: Júnior = programador iniciante; Pleno = programador de experiência intermediária; Sênior = programador experiente.

Engenheiros de *software* relatam que a programação em pares é mais útil para projetar o *software* e tarefas complexas de programação. (VANHANEN et al., 2007)

Beck (2012) baseado em sua experiência de décadas com a programação em pares, publicou em seu perfil @KentBeck na rede social Twitter, que a programação em pares funciona melhor quando existe um grande espaço de incerteza na busca pela solução de problemas e soluções, quanto mais próximo da solução do problema, menos a técnica ajuda.

Sison (2009) observou em dois experimentos realizados com quarenta e oito estudantes do terceiro ano de ciências de computação, que há melhora da qualidade (foi medida através da densidade de defeitos) ao usar a programação em pares e quando o projeto é suficientemente grande ou complexo para permitir a colaboração entre os programadores, a perda de produtividade é reduzida.

Gênero, confiança e satisfação dos programadores

A programação em pares estimula a motivação e satisfação dos programadores (PADBERG et al., 2004). Ela melhora a qualidade do código, aumenta a confiança e satisfação dos programadores (observou-se aumento de 24% para mulheres e 15% para homens) e reduz a evasão de estudantes dos cursos de computação, pesquisa coletou dados de 554 estudantes da universidade Santa Cruz na California-USA. (MCDOWELL et al., 2006).

Choi (2013) conduziu um experimento para avaliar a relevância do gênero no contexto da programação em pares, participaram 128 estudantes, sendo 93 homens e 35 mulheres. Observa-se no resultado que não há diferença na habilidade de programação e resolução de problemas, porém, ocorre diferença significativa na comunicação e compatibilidade do par, pares do mesmo gênero, homem-homem e mulher-mulher, apresentaram maior nível de comunicação e compatibilidade. Pares do mesmo gênero, pode ser uma diretriz, na hora de realizar a seleção dos pares.

Mecanismos para melhorar o desempenho da programação em pares

Wray (2010) apresentou quatro mecanismos para melhorar o desempenho da programação em pares: conversar, avisar mais detalhes, lutar contra práticas ruins de programação e compartilhar e reconhecer competências.

A conversa entre os programadores é importante para o bom desempenho da técnica, parte da eficácia da programação em pares é presumivelmente devida à interação entre os programadores. Quando os programadores falam sobre o

problema no momento da dúvida, encontram mais rapidamente a solução, e consequentemente há o aumento da produtividade**.**

Nota-se mais detalhes ao programar em par, portanto, é importante avisar o par, pois talvez ele não tenha notado o mesmo ponto, pois a capacidade de percepção do ser humano depende do objetivo da busca, há um ponto cego na atenção humana em relação ao inesperado, o ser humano tem dificuldade de notar algo que não estava esperando, que não estava concentrado naquele ponto. Isso ajuda a explicar o motivo da rotação do par evitar a fadiga. A fadiga diminui a produtividade, e trocar a pessoa do par que digita no teclado durante o expediente pode aumentar a produtividade. É importante levar em consideração que programadores iniciantes aumentam sua produtividade quando seu par é mais experiente.

Lutar contra práticas ruins de programação aumenta o desempenho**,** a pressão do par contribui para melhorar a qualidade da programação, porém, deve-se evitar considerar que o trabalho num trecho de código está concluído simplesmente porque executou sem erro de sintaxe. É importante buscar seguir os padrões de programação e programar da melhor maneira possível para que o código não se torne ruim e desorganizado, desta maneira evita-se retrabalhos e aumenta a produtividade. Compartilhar e reconhecer competências, promove a disseminação do conhecimento**.** A descoberta de quem são os especialistas em determinados assuntos e compartilhar o conhecimento pode aumentar a produtividade. Wray (2010) sugere ainda refazer a pesquisa de Dyba et al. (2007) considerando seus quatro mecanismos e pensar em novos critérios e método para mensurar eficientemente o desempenho, levando em consideração estes mecanismos.

É realizada uma adaptação na programação em pares por Swamidurai e Umphress (2015) onde o par não trabalha lado a lado em todas as fases de desenvolvimento de software; em vez disso, a dupla começa projetando juntos, se separaram durante a implementação, em seguida, juntam-se novamente para o teste. Dois estudos empíricos foram realizados na primavera de 2009 e no verão de 2012 para validar a programação em par invertida contra programação em pares tradicional. Os resultados deste experimento são favoráveis à programação par invertida e não suportam a alegação de que a programação em pares, em geral reduz a duração de desenvolvimento de *software* ou custo total de desenvolvimento de *software*.

Posse do teclado

É importante o par revezar a posse do teclado para manter ambos comprometidos (VANHANEN; KORPI, 2007); (WILLIAMS; KESSLER, 2003). Quando o par tem nível diferente de experiência, o programador mais experiente tende a dominar a posse do teclado. Os engenheiros de software parecem mais comprometidos na posse do teclado. Para promover o comprometimento de ambos, é sugerido o uso de dois teclados e dois mouses. (CHONG; HURLBUTT, 2007); (FREUDENBERG; ROMERO; BOULAY, 2007); (HÖFER, 2008).

Plonka, Sharp e van der Linden (2012) publicaram um estudo sobre programação em pares com programadores profissionais e observaram que dentre os pares, quem não estava na posse do teclado, muitas vezes tinham razão para separar-se, dependendo da atividade de quem esta na posse do teclado, devido as interrupções ou porque eles dividiram trabalho a ser feito em paralelo. É observado que essa separação não é necessariamente um problema.

Programação em pares distribuída

A programação em pares é uma prática que pode ser implementada de forma distribuída. O trabalho de Bandukda et al.(2010) concluiu que a programação em pares distribuída, onde os pares estão em dependências físicas diferentes e comunicam-se por voz e software de mensagens de texto, pode ser usada para aumentar a produtividade do desenvolvimento e a qualidade do produto.

Um trabalho de Williams e Stout (2008) descreveu algumas modificações nas práticas ágeis adotadas por uma empresa para reduzir os problemas no desenvolvimento de software distribuído entre America do Norte e Europa. A criação do papel do “usuário ágil”. Assim como o “analista de negócio suplente do cliente” recomendado por Hildebrand et al. (2008) e a criação de “salas virtuais de conferência”, para resolver problemas de ineficiência de ferramentas de comunicação e de disciplina durante as reuniões. Concluíram que é possível distribuir e escalar equipes ágeis, mas mantê-las pequenas e juntas é a melhor alternativa.

Estácio e Prikladnicki (2015) realizaram uma revisão sistemática sobre a programação em pares distribuída e concluiu-se que há a necessidade de mais estudos na indústria de *software* sobre a programação em pares distribuída; há uma boa oportunidade para pesquisas neste tema; há uma tendência no uso e pesquisa da programação em pares distribuída no ensino de programação. Há a necessidade

de investigar os efeitos da programação em pares distribuída em profissionais. O nível de dispersão das equipes foi pouco contextualizada nos trabalhos pesquisados. Há oportunidade para investigação dos efeitos da coordenação, comunicação e diversidade cultural na programação em pares distribuída. Há a necessidade de explorar mais diretrizes (*guidelines)* para implementação da programação em pares distribuída na industria de *software.* Há oportunidade para explorar o projeto, a implementação e as ferramentas voltadas à programação em pares distribuída. Há oportunidade para investigar a programação em pares distribuída com outras formas de programação colaborativa.

Preceptores de desempenho e cultura organizacional

Segundo Hannay et al. (2010), a personalidade pode ser um indicador válido para o desempenho da equipe no longo prazo. No entanto, não é encontrado indícios fortes que a personalidade afeta o desempenho ou benefícios da programação em pares de uma forma consistente, especialmente quando incluindo indicadores relativos à perícia, a complexidade da tarefa e do país. No curto prazo, vale a pena para indústria e pesquisadores, concentrarem-se em outros preceptores de desempenho, incluindo experiência e complexidade da tarefa. Desperta-se atenção para os fatores que podem ser relevantes, tais como a aprendizagem em equipe, motivação da equipe, e habilidade de programação. Em contraste com traços de personalidade, que são fixados em uma pessoa, são fatores maleáveis, tais como: aprendizagem, motivação, e empenho para realizar benfeitorias ao *software* produzido.

Plonka e van der Linden (2012) identificaram em quatro empresas, aspectos que dificultam a adoção da programação em pares. Eles descrevem que estes aspectos estão relacionados com questões da organização, como a cultura organizacional e estilo de gestão. Como consequência, a cultura organizacional é considerada fundamental para a ampliação da adoção da programação em pares entre as equipes destas quatro empresas estudadas.

Companheirismo

Foi realizado por Plaue et al. (2013) um estudo experimental de programação em pares contra programação individual. O resultado mostra que a programação em pares estimula o companheirismo. É um fator de proteção para conservação e diminuição da evasão de mulheres e programadores novatos nos cursos de ciências

da computação. Pode ser uma intervenção de baixo custo para a redução do *turn-over* em equipes de *software*.

Interrupções e aprendizagem cognitiva

Sillitti et al. (2012) observaram que as pessoas mudam de forma relevante seus hábitos de trabalho quando trabalham em pares e tendem a concentrar-se mais em atividades produtivas. As sessões de trabalho são significativamente mais longas e ininterruptas e eles concentram-se mais nas tarefas, diminuindo distrações como e-mails, navegadores de internet e programas de bate-papo. Ao mesmo tempo é encorajada a discussão, melhorando a solução criada e ajudando o par a ter uma visão mais clara do problema a ser resolvido.

Jones e Fleming (2013) fizeram um estudo qualitativo de 7 pares (14 estudantes), durante tarefas de depuração de código, enfatizaram questões como ensinamento do parceiro, contribuições do parceiro que não está na posse do teclado e o impacto das interrupções do parceiro. Observa-se nos resultados que todos os pares apresentaram episódios de ensinamento do parceiro, a maioria sobre programação, os parceiros que não estavam na posse do teclado, contribuíram com várias ideias que foram implementadas pelo par de posse do teclado sem discussão, não foi apresentado nenhum indício que as interrupções do parceiro foram prejudiciais ao fluxo do trabalho.

Plonka, Sharp, van der Linden e Dittrich (2014) observaram a programação em pares como estratégia para transferência de conhecimento entre programadores seniores e novatos, este é um dentre vários métodos de ensino usados na aprendizagem cognitiva. O desenvolvedor sênior verbalizar seu ensinamento e o novato ser apoiado pelo sênior quando ele está desenvolvendo uma tarefa são métodos de ensino usados na aprendizagem cognitiva e foram observados na pesquisa. Reflexão (comparar sua solução com a de outros) e a exploração (tarefas para o novato avançar no aprendizado) também são métodos de ensino usados na aprendizagem cognitiva e não foram observados, pois, as necessidades da empresa de produzir código são priorizadas em detrimento ao aprendizado, bem como, a verbalização do novato durante o aprendizado não é estimulada nas empresas pesquisadas, porém, melhoraria o processo de aprendizado do novato. Embora os seniores façam um esforço adicional para transferir conhecimento ao novato, ao aumentar a consciência sobre práticas de transferência de conhecimento, gera oportunidade de aprendizado para o sênior.

Relação com projeto de interface humano computador

Seyamn e Mccrickard (2015) pesquisaram com estudantes de ciências da computação, a programação em pares e requisitos de interface humano computador e recomendam para os estudantes iniciantes em programação em pares que as duplas não sejam formadas aleatoriamente, é melhor os estudantes escolherem seus pares ou o par ser formado com base em suas notas na aula de programação. Os estudantes devem ser encorajados a gastar um tempo no projeto e como será feito o trabalho. Introduzir a qualidade como um requisito básico, ao invés de como bônus. Enfatizar a regra de falar e ouvir. É melhor avaliar por meio de um desafio de programação. Os estudantes devem ser explicitamente questionados acerca dos contextos e cenários dos usuários. Eles afirmam que a programação em pares contribui para uma maior interação entre os desenvolvedores, o que permite alcançar um melhor projeto de interface humano computador e melhor experiência para o usuário ao usar o *software* por eles desenvolvido.

A seguir, o Quadro 2 sintetiza os pontos fortes e fracos da programação em pares observados pelos trabalhos de seus respectivos autores em ordem cronológica. Os trabalhos representam os principais autores referenciados para viabilizar um estudo terciário como forma de embasar uma nova técnica que combine os pontos fortes e evite os pontos fracos.

Quadro 2 – Pontos fortes e fracos relacionados à programação em pares

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
WEINBERG	1971	Programação sem ego.	
BROOKS	1975	Entre 1953 e 1956, Brooks e seu par escreveram 1500 linhas de código, que executaram sem nenhum erro na primeira tentativa.	É apenas um relato realizado com apenas dois programadores iniciantes.
CONSTANTINE	1995	Redução do número de defeitos.	
COPLIEN; SCHMIDT	1995	Publicação de um padrão chamado desenvolvimento em pares. Pessoas às vezes sentem que apenas podem resolver um problema se tiverem ajuda. Alguns problemas são maiores do que qualquer indivíduo.	

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
NOSEK	1998	Melhora a qualidade do código e algoritmo.	Aumenta o esforço no desenvolvimento, pois aumenta o número de horas de trabalho ao somar-se as horas de trabalho do par.
BECK	1999	Melhora a qualidade do produto de *software.*	
COCKBURN; WILLIAMS.	2000	Comunicação e colaboração constantes geram confiança	
WILLIAMS; KESSLER; CUNNINGHAM; JEFFRIES	2000	Aumento da satisfação e confiança entre os programadores. Redução do número de erros e retrabalho. É um processo barato de revisão informal; melhora a qualidade do código desenvolvido e nível técnico da equipe. O número de erros evitados pela inspeção informal realizada pela programação em pares é tal que, menos tempo é gasto na reparação de *bugs* descobertos durante o processo de testes.	Trabalho realizado com programadores iniciantes
WILLIAMS; KESSLER	2000	Disseminação do conhecimento e compartilhamento de técnicas e competências entre os membros da equipe.	
COCKBURN; WILLIAMS	2001	A produtividade, ao usar a programação em pares, parece ser comparável à de duas pessoas trabalhando de forma independente. As razões sugeridas são que programando em pares discutem o sistema antes de desenvolvê-lo, então provavelmente ocorrerá menos erros e retrabalho. Estimula a refatoração.	Trabalho realizado com programadores iniciantes

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
SMITH; MENZIES	2002	É vantajoso para NASA em projetos relativamente pequenos; quando existe uma abundância de desenvolvedores e se for necessário um período de desenvolvimento curto.	Perda de produtividade em projetos médios/grandes e sem abundancia de desenvolvedores
WILLIAMS; KESSLER	2003	A inclusão de um novo membro na equipe que não conhece o sistema cria o problema do primeiro contato com o código. A programação em pares contribui para resolução deste problema, pois o novo membro da equipe pode trabalhar em par com alguém mais experiente.	Engenheiros de software parecem ficar mais comprometidos na posse do teclado. Revezar a posse do teclado, pode ajudar a preservar o comprometimento de ambos desenvolvedores.
PADBERG, MÜLLER	2003	Quando a pressão do mercado é forte, acrescentar desenvolvedores e formar mais pares, pode acelerar o projeto e aumentar o seu valor para o negócio, apesar de aumentar também o custo.	Aumento do custo no desenvolvimento de *software.*
PADBERG, MÜLLER	2004	Estimula a motivação e satisfação dos programadores.	
BECK; ANDRES	2004	Disseminação do conhecimento, resolução do problema de um único desenvolvedor deter o conhecimento sobre uma parte do sistema e elevação do nível técnico da equipe.	
PARRISH; SMITH; HALE, D; HALE, J	2004	Há alguns benefícios de qualidade, mas estes não compensam totalmente a perda da produtividade. No entanto, a partilha de conhecimento que acontece durante a programação em pares é muito importante, pois reduz os riscos globais para um projeto no caso de algum membro da equipe sair e isso pode justificar o seu uso.	Significativa perda de produtividade em comparação à programadores trabalhando sozinhos quando os programadores do par são experientes.

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
SLATEN; WILLIAMS; BERENSON.	2004	Promove a confiança entre os programadores.	
COPLIEN; HARRISON	2005	Projetos compatíveis com o emparelhamento de trabalhar em conjunto; desta forma podem produzir mais do que a soma dos dois individualmente.	
HULKKO; ABRAHAMSSON	2005	Observaram em quatro estudos de caso a eficácia da programação em pares na disseminação de conhecimento e que ela é mais útil em tarefas complexas.	Quando a complexidade da tarefa for baixa é menos indicado o uso da programação em pares.
MCDOWELL; WERNER; BULLOCK; FERNALD	2006	Ela melhora a qualidade do código, aumenta a confiança e satisfação dos programadores(aumentou em 24% para mulheres e 15% para homens) e diminui a evasão de estudantes dos cursos de computação.	Pesquisa realizada com 554 estudantes da universidade Santa Cruz na California-USA
VANHANEN; LASSENIUS; MÄNTYLÄ	2007	A programação em pares é mais útil para projetar o *software* e tarefas complexas de programação	
VANHANEN; KORPI	2007	É importante o par revezar a posse do teclado para manter ambos programadores comprometidos.	
CHONG; HURLBUTT	2007	Os engenheiros de software parecem mais comprometidos na posse do teclado. Para promover o comprometimento de ambos, é sugerido o uso de dois teclados e dois mouses.	Quando o par tem nível diferente de experiência, o programador mais experiente tende a dominar a posse do teclado.
FREUDENBERGROMERO; BOULAY	2007	Os engenheiros de software parecem mais comprometidos na posse do teclado. Para promover o comprometimento de ambos, é sugerido o uso de dois teclados e dois mouses.	

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
ARISHOLM; GALLIS; DYBA; SJOBERG	2007	Há alguns benefícios de qualidade, mas estes não compensam totalmente a perda da produtividade.	Perda de produtividade e não é vantajosa para programadores experientes e nem para programadores plenos quando a tarefa possa ser considerada simples.
DYBA; ARISHOLM; SJOBERG; HANNAY; SHULL	2007	Melhora da qualidade e redução do número de defeitos,	Perda de produtividade e não é vantajosa para programadores experientes e nem para programadores plenos quando a tarefa possa ser considerada simples.
HÖFER	2008	Os engenheiros de software parecem mais comprometidos na posse do teclado. Para promover o comprometimento de ambos, é sugerido o uso de dois teclados e dois mouses.	
BEGEL; NAGAPPAN	2008	Estimula a disseminação do conhecimento, reduz a quantidade de defeitos e gera *software* com mais qualidade.	
SISON	2009	Melhora a qualidade (foi medida através da densidade de defeitos) ao usar a programação em pares e quando o projeto é suficientemente grande ou complexo para permitir a colaboração entre os programadores, a perda de produtividade é reduzida.	
HANNAY; ARISHOLM; ENGVIK; SJOBERG	2010	Não é encontrado indícios fortes que a personalidade afeta o desempenho ou benefícios da programação em pares de uma forma consistente, especialmente quando incluindo indicadores relativos à perícia, a complexidade da tarefa, e do país.	Desperta-se atenção para os fatores que podem ser relevantes e estão sob investigação em curso, tais como a aprendizagem em equipe, motivação da equipe, e habilidade de programação.

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
BANDUKDA; NASIR	2010	A programação em pares é uma prática que pode ser implementada de forma distribuída. A programação em pares distribuída pode ser usada para aumentar a produtividade do desenvolvimento e a qualidade do produto.	
WRAY	2010	Quatro mecanismos para melhorar o desempenho da programação em pares: conversar, avisar mais detalhes, lutar contra práticas ruins de programação e compartilhar e reconhecer competências.	Wray sugere refazer a pesquisa de Dyba et al. (2007) considerando seus quatro mecanismos e pensar em novos critérios e método para mensurar eficientemente o desempenho, levando em consideração estes mecanismos.
SILLITTI; SUCCI, VLASENKO	2012	Mudança relevante de hábitos de trabalho e os desenvolvedores tendem a concentrar-se mais em atividades produtivas.	
PLONKA; VAN DER LINDEN	2012		Requer ajustes na cultura e estilo de gestão da organização.
PLONKA; SHARP; VAN DER LINDEN	2012	Dentre os pares, quem não estava na posse do teclado, muitas vezes tinham razão para separar-se, dependendo da atividade de quem esta na posse do teclado, devido as interrupções ou porque eles dividiram trabalho a ser feito em paralelo. É observado que essa separação não é necessariamente um problema.	
CHOI	2013		Pares do mesmo gênero, homem-homem e mulher-mulher, apresentaram maior nível de comunicação e compatibilidade. Pares do mesmo gênero, pode ser uma diretriz, na hora de fazer a seleção dos pares.

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
JONES; FLEMING	2013	Observa-se nos resultados que todos os pares apresentaram episódios de ensinamento do parceiro, a maioria sobre programação, os parceiros que não estavam na posse do teclado, contribuíram com várias ideias que foram implementadas pelo par de posse do teclado sem discussão, não foi apresentado nenhum indício que as interrupções do parceiro foram prejudiciais ao fluxo do trabalho.	Trabalho realizado com apenas 14 programadores iniciantes/estudantes
PLONKA; SHARP; VAN der LINDEN; DITTRICH	2014	O desenvolvedor sênior verbalizar seu ensinamento e o novato ser apoiado pelo sênior quando ele está desenvolvendo uma tarefa são métodos de ensino usados na aprendizagem cognitiva e foram observados na pesquisa com sucesso.	Reflexão (comparar sua solução com a de outros) e a exploração (tarefas para o novato avançar no aprendizado) também são métodos de ensino usados na aprendizagem cognitiva e não foram observados
SWAMIDURAI; UMPHRESS	2014	Aumento da produtividade quando há rotação dos pares na equipe	Programação em pares estática é menos produtiva que a dinâmica
SWAMIDURAI; UMPHRESS	2015	Dois estudos empíricos foram realizados na primavera de 2009 e no verão de 2012 para validar a programação em par invertida contra par programação tradicional. Os resultados deste experimento ponto a favor da metodologia de programação par invertida e não suportam a alegação de que a programação em pares, em geral reduz a duração de desenvolvimento de software ou custo total de desenvolvimento de software.	Programação em pares é menos produtiva que a Programação em pares invertida

Continua na próxima página

AUTOR(ES)	ANO	PONTOS FORTES	PONTOS FRACOS
ESTÁCIO; PRIKLADNICKI	2015		Há a necessidade de mais estudos na indústria de *software* sobre a programação em pares distribuída. Há a necessidade de explorar mais diretrizes (*guidelines)* para implementação da programação em pares distribuída na industria de *software.*
DU; OZEKI; NOMIYA; MURATA; ARAKI	2015	Eficaz para melhorar a comunicação em um exercício básico de C	
SEYAM; MCCRICKARD	2015	Contribui para uma maior interação entre os desenvolvedores, o que permite alcançar um melhor projeto de interface humano computador e melhor experiência para o usuário ao usar o *software* por eles desenvolvido.	

Fonte: Elaborado pelo autor

2.1.1 Resumo dos pontos fortes e fracos

A revisão da literatura relacionada à programação em pares ilustrada no Quadro 2, evidencia a diminuição do número de defeitos, a disseminação do conhecimento, o aumento da qualidade do produto de *software,* existe um problema de produtividade associado à técnica, apenas alguns trabalhos levaram em conta a complexidade da tarefa e a experiência do programador e, observaram que a técnica funciona melhor para programadores novatos e tarefas complexas.

Para promover o comprometimento, é sugerido o uso de dois teclados e dois mouses. Para melhorar o desempenho é sugerido os mecanismos: conversar, avisar mais detalhes, lutar contra práticas ruins de programação e compartilhar e reconhecer competências.

Observa-se no Quadro 2, o aumento da produtividade quando há rotação dos pares na equipe. Observa-se também, o aumento da produtividade quando a programação em pares é invertida, onde o par não trabalha lado a lado em todas as fases de desenvolvimento de software; em vez disso, a dupla começa projetando juntos, se separaram durante a implementação, em seguida, juntam-se novamente para o teste.

2.2 Programação em Grupo (*Mob Programming*)

A ideia de programar em grupo originou-se de reuniões de almoço de desenvolvedores, originalmente executado em um formato de apresentação, onde um membro da equipe apresentava um código que conhecia. Esperava-se proporcionar maior oportunidade para os desenvolvedores trabalharem juntos, permitindo a troca de informações sobre técnicas de programação, encorajar mudanças, *feedbacks,* disseminar conhecimento da arquitetura de *software* do projeto, fortalecer o senso de equipe. (HOHMAN E SLOCUM, 2001)

Programação em grupo é uma técnica onde toda a equipe participa em torno de uma estação de trabalho com uma única pessoa na posse do teclado, mouse e computador. É programação em par com mais de duas pessoas. (ZUILL, 2012)

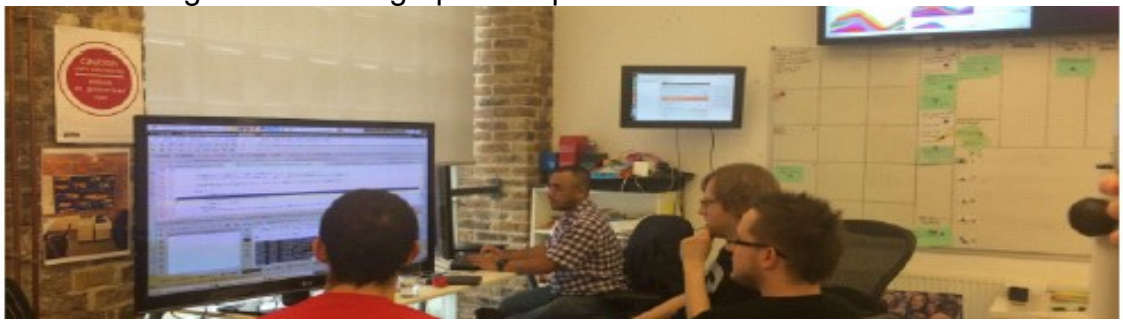
Hohman e Slocum (2001) definem o *mob programming* (programação em grupo) como sendo a refatoração de um trecho de código em grupos maiores do que duas pessoas. Conduziram um experimento em seu time de desenvolvimento, composto por cerca de dez profissionais, para averiguar se os benefícios da programação em pares são preservados ao programar em grupo de mais de duas pessoas em um único computador. No experimento deles praticou-se a programação em grupo oito vezes, relatou-se dificuldade para seguir a própria diretriz. Por exemplo, uma boa preparação, tem um efeito forte sobre a qualidade de uma apresentação. No entanto, apenas uma minoria dos casos esta preparação foi realizada adequadamente. Isso é apontado como uma fraqueza, a partir do ponto de vista do método XP. Neste experimento são testados dois formatos de *mob programming*, um formato com toda a equipe num mesmo grupo e apenas um projetor e outro onde a equipe é dividida em dois grupos, e cada grupo menor usa seu próprio projetor. Um projetor é um pouco mais fácil de gerir logisticamente. No entanto, porque o grupo é composto por cerca de dez desenvolvedores, tendo apenas um ponto de foco é difícil incluir todos no processo. Com dois projetores, é possível experimentar vários diferentes formatos, em geral, mais envolvente. Por exemplo, ao se dividir em duas equipes, uma possibilidade é cada uma fazer a sua própria refatoração de código e comparar/discutir com a feita pela outra equipe, este formato é observado o mais bem sucedido. Outro formato é enquanto uma equipe refatora, a outra equipe trabalha nos testes, este formato sofre do problema de que para refatorar, é necessário os testes prontos e uma vez que os testes estejam prontos, muitas vezes é necessário esperar para que a refatoração termine, pois

guia a escrita dos testes. Uma equipe acaba esperando a outra. Em ambas possibilidades, com dois projetores, os pontos de foco envolvem mais pessoas, há compartilhamento dos papéis, envolvendo mais amplamente os membros do grupo.

É observado no experimento de Hohman e Slocum (2001) que as pessoas acham difícil manter o foco, especialmente aqueles que não estão na posse do teclado. Mais de dois programadores evidenciaram a propensão para a dispersão da atenção, terem outras conversas não relacionadas, a menos que todos na sala estejam envolvidos na prática, senão, o grupo inevitavelmente torna-se incontrolável. São feitas críticas mistas, os inegáveis benefícios são indiretos, como: a melhora da comunicação e cobertura dos testes. É sugerido duas melhorias: completar a preparação necessária e estabelecer um treinador para supervisionar indiretamente o trabalho. A equipe sugere alternar a programação em grupo com outros formatos.

Na empresa onde Wilson (2015) trabalha, estão constantemente tentando melhorar as práticas do eXtreme Programming. No segundo semestre de 2014, mudaram da Programação em Par em todo o código de programação produzido para *Mob Programming* com toda a equipe, influenciados pela apresentação de Zuill (2014), assemelhava-se ao estilo *Randori* de *Coding Dojos* (ROOKSBY et al., 2014), que já usam durante as sessões de codificação para aprender novas tecnologias. A equipe declarou que todas as sextas-feiras seriam em grupo (*Mob Fridays).* Colocam um quadro branco como uma divisória, isolando a equipe do resto do pessoal da empresa, na tentativa de aumentar o envolvimento da equipe. Colocar toda equipe na frente de um monitor é um problema para eles, pois a área de desenvolvimento é aberta. A solução, conforme mostra a Foto 1, foi colocar os monitores grandes que eles já usam para programar em par em conjunto com um monitor de 50 polegadas espelhado no monitor grande da estação de trabalho, na intenção, de quem estiver na posse do teclado possa ter a visão além do seu monitor, da equipe inteira.

Foto 1 – Programando em grupo de 4 pessoas



Fonte: Wilson (2015)
Legenda: Ao fundo está o programador que está na posse do computador

A equipe que Wilson (2015) faz parte, relata uma maior produtividade com dois monitores, ao invés, de ter apenas um e todos olharem para o mesmo monitor e apenas passar o teclado entre os desenvolvedores.

O uso de dois monitores facilita a conversa entre quem esta na posse do teclado e os observadores. Aumentou a confiança usando a programação em grupo em comparação a programação em pares que era adotada antes, observaram que fortaleceu a equipe e o aprendizado em conjunto.

É adotada a rotação da posse do teclado a cada 5 minutos, o que significa num grupo de 4-6 pessoas um intervalo de 15-25 para pilotar novamente o teclado. É observada a importância de um rápido e atualizado conjunto de testes automatizados. É pretendido dar sequência na adoção do *mob programming* para averiguar realmente se é verídica a alegação de aumento da produtividade devido a melhora do *throughput* gerado a partir do *One Piece Flow*, elemento fundamental da abordagem *Lean*, onde é processada uma unidade de ordem do cliente por vez, deve-se processar somente o que o cliente quer, na quantidade e quando ele quiser. (KAIZENWORLD, 2015)

É relatado por Wilson (2015) a preocupação dos membros não desenvolvedores, dentre eles o gerente de produto, acerca da dificuldade de convencer a empresa que colocar toda equipe trabalhando junta numa história é mais produtivo e eficaz do que ter pares trabalhando em histórias diferentes.

A equipe tomou a decisão que se o time inteiro necessitasse ser interrompido, se daria após a completa rotação do time. São observados dois problemas, que limitam a eficácia do *mob programming*: o efeito de personalidades dominantes dentro do grupo e o aparecimento do *Groupthink,* um bem documentado efeito psicológico, no qual segundo Janis (1982), a tendência para desenhar uma percepção enviesada da situação, devido ao excesso de confiança do grupo, tanto em suas habilidades técnicas e quão bem perceberão os eventos futuros. Juntos este dois problemas tornam a programação em grupo menos útil que a programação em pares.

A disseminação do conhecimento operacional foi estimulada e foi observado que funcionou bem para os códigos críticos e complexos. (WILSON, 2015)

Devido aos problemas de *Groupthink* e personalidades dominantes em sua equipe, decidiram não usar sempre *mob programming,* porém, é uma técnica útil em

situações particulares e para resolver problemas específicos, mais uma das muitas ferramentas para a equipe. (WILSON, 2015)

Conforme observado por Wilson e Weber (2015) a programação em grupo certamente não resolve os problemas de um código mal escrito ou processo de desenvolvimento de *software* ruim. Pode inclusive amplificar disfunções da equipe, bem como os pontos fortes.

Eles relatam que a programação em grupo ajudou sua equipe na resolução dos problemas mais complexos e críticos, também ajudou a envolver colegas de outras áreas, além do desenvolvimento de software nas sessões de programação em grupo, incluindo especialistas em produtos, infraestrutura e experiência do usuário. (WILSON E WEBER, 2015)

Segundo Zuill (2015) a equipe decidiu usar o *Mob Programming* e software de alto valor é entregue há mais de três anos. É uma abordagem para equipe inteira fazer todo o trabalho em conjunto, incluindo programação, projeto, testes, e trabalho com o cliente (sócios, *product owner*, usuário, etc). Expande-se a natureza de todo trabalho feito pela equipe, além do planejamento, retrospectivas, e um *standup* diário ou outra reunião. Este é um passo evolutivo para a programação em pares, comunicação presencial, o alinhamento da equipe, colaboração e conceitos de equipe auto-organizável da abordagem ágil de desenvolvimento de software.

2.2.1 Pontos fortes e fracos

Em relação aos pontos fracos, Hohman e Slocum (2001) relatam que é difícil manter o foco, especialmente para aqueles que não estão na posse do teclado. Wilson (2015) cita dois problemas, o efeito de personalidades dominantes dentro do grupo e aparecimento do *Groupthink.*

Dentre os pontos fortes, Zuill (2015) destaca que fortalece a comunicação presencial; alinhamento da equipe; colaboração e conceitos de equipe auto-organizável. Wilson e Weber (2015) observam que a disseminação do conhecimento operacional foi estimulada e a técnica funcionou bem para os códigos críticos e complexos.

Há dois pontos de convergência na conclusão de Hohman e Slocum (2001); Wilson (2015); Wilson e Weber (2015), ao usar dois projetores ou monitores a técnica funciona melhor e é vantajoso alternar o uso da programação em grupo com outras técnicas.

2.3 Revisão de Código (*Code Review*)

A questão da revisão da literatura da revisão de código em pares é: Quais são os pontos fortes e fracos observados e/ou constatados nos últimos dois anos sobre revisão de código moderna em pares? As palavras chaves utilizadas para pesquisa dos artigos abrangem: *code review*, *modern code review, pair review, pair code review* e *modern pair review*.

Segundo Carver et al. (2015) os dois pontos principais de engenharia de *software* para este ano são: o primeiro é a colaboração entre indústrias de *software* e universidades; o segundo é a revisão de código moderna, que é uma versão reduzida das tradicionais inspeções de código e observa-se o aumento do número de pesquisas sobre revisão de código moderna, devido a sua importância para a indústria de *software.*

Morales el al. (2015) examinaram o impacto de práticas modernas de revisão de código na qualidade do projeto (*design*) e com base na evidência empírica coletada em três grandes projetos *open sources* (*Qt, VTK e ITK*) concluíram que contribui para melhora da qualidade do projeto de *software.*

A análise baseada nos dados de 25 mil desenvolvedores da Microsoft, propicia uma observação interessante, o principal objetivo de uma revisão de código costuma ser identificar defeitos, porém, apenas 15% dos comentários dos revisores é sobre defeitos. Melhorar a manutenibilidade do código é um comentário presente em mais de 50% do total de comentários dos revisores. (CZERWONKA et al., 2015)

Outra observação interessante é o número máximo ideal de 20 arquivos por revisão, a partir deste ponto de saturação a qualidade dos comentários feitos pelo revisor, diminui. (CZERWONKA et al., 2015)

Os resultados na Microsoft mostram que os desenvolvedores precisam de 6 meses a 1 ano de experiência com o código fonte para se tornarem revisores eficientes. Ainda não há, uma completa compreensão sobre o *workflow* ideal e atualmente não são tão eficazes e não se pode negligenciar aspectos sociais da revisão de código. (CZERWONKA et al., 2015)

Swamidurai, Dennis e Kannan (2014) comparam duas técnicas de revisão de código colaborativas, a programação em pares tradicional e a revisão de código em pares, que melhoram a qualidade de programas, são alternativas as inspeções formais e rigorosas que costumam remover 90% dos defeitos antes de ser

executado o primeiro caso de teste. O conceito tradicional de programação em pares tem o requisito do par estar sentado lado a lado, o que é um obstáculo em muitos projetos de desenvolvimento. Revisões de código em pares, no entanto, têm se mostrado tão eficazes quanto programação em pares e são um ajuste melhor para muitos *softwares* que estão sendo desenvolvidos colaborativamente de forma assíncrona e na nuvem. Realizaram um experimento utilizando a técnica de revisão de código em pares, em comparação com a técnica de programação em pares tradicional no contexto em que é adotado o *Test Driven Development*. O resultado mostra que programas com qualidade equivalente podem ser produzidos com um custo 28% menor em relação à programação em pares.

Zhang et al. (2015) propõe a revisão de código interativa para mudanças sistemáticas, que são mudanças similares relatadas para vários contextos, este recurso de exibir as mudanças é muito útil para o revisor do código, porém, as vezes, os revisores deparam-se com questões como: em quais outros ponto do código há alterações similares?

Mesmo as ferramentas populares específicas para revisão de código como: *Phabricator* (2015)*, Gerrit* (2015)*, Collaborator* (2015) e *Codeflow* (2015), computam mudanças, por arquivo, o que obriga a leitura de linhas de alterações por arquivo, então para resolver este problema é criado o *Critics* (2015)*,* uma abordagem interativa para revisão de código para mudanças sistemáticas, consiste em um *plug-in* para *Eclipse* para detecção de mudanças sistemáticas de forma automática, similar ao *Eclipse/Git/SVN Diff* ou *show differences* da Microsoft®*.*

Primeiro escolhe a região da mudança, depois permitem ao desenvolvedor customizar o conteúdo do *template* da mudança e executam o *plug-in* com base no *template* para detectar possíveis anomalias no código. Os resultados de Zhang et al. (2015) indicam que o *Critics* (2015) melhora a acurácia e produtividade das mudanças sistemáticas durante o processo de revisão de código.

2.3.1 Pontos fortes e fracos

Revisão de código moderna em pares é observada como um ajuste melhor ao desenvolvimento colaborativo assíncrono e na nuvem e contribui para a melhora da qualidade do projeto (*design*) de *software.* Ainda não há, uma completa compreensão sobre o seu *workflow* ideal e atualmente não são tão eficazes e não se pode negligenciar os aspectos sociais da revisão de código.

2.4 Engenharia Simultânea (*Concurrent Engineering*)

A Engenharia Simultânea ou Concorrente, é uma maneira de abordar o desenvolvimento de produtos, desde a revolução industrial foi crescente a necessidade de uma maneira de se pensar em como desenvolver novos produtos ou melhorar os processos para que se produza mais. Até a primeira metade do século XX os projetos de desenvolvimento de produtos abrangiam menos áreas, menos interesses e ,dessa maneira, podemos dizer que os projetos era menos complexos. Com o progresso científico e tecnológico os projetos foram se tornando cada vez mais complexos e se tornando cada vez mais interdisciplinares,consequentemente, aumentando o número de pessoas envolvidas em sua concepção. Inicialmente produtos eram desenvolvidos de uma maneira sequencial. Com o aumento de sua complexidade e da competição entre as organizações produtoras, houve a necessidade de encurtar o tempo de desenvolvimento e grupos multidisciplinares passaram a ser utilizados para realizar o desenvolvimento do produto e dos seus processos do ciclo de vida, simultaneamente. Essa abordagem foi chamada de engenharia simultânea. (Pires e Loureiro, 2010)

O principal objetivo é aumentar a competitividade dos produtos aumentando a sua qualidade e ao mesmo tempo reduzindo os custos e o tempo do projeto. É fundamental para isso, o trabalho em equipe (Deboni et al., 1994).

O trabalho colaborativo, as equipes multidisciplinares e o apoio de *softwares* para orquestrarem o trabalho simultâneo são fundamentais para a competitividade. (Deboni et al., 1995)

Segundo Bennett et al. (1995) o principal objetivo da Engenharia Simultânea é a diminuição do tempo desde o pedido até a entrega, para um novo produto, com custo mais baixo e maior qualidade. Isto é alcançado pelo desenvolvimento paralelo, ao invés de sequencial, das diferentes etapas que compõem o projeto do produto, com o emprego de times ou equipes multidisciplinares. Engenharia Simultânea usa tecnologia para atingir seus objetivos.

Hauptman e Hirji (1996) colocam que dentre os princípios da Engenharia Simultânea estão objetivos comuns, completa visibilidade dos parâmetros de projeto, consideração mútua de todas as decisões, colaboração para resolver conflitos, equipes de trabalho, melhoria contínua. O desenvolvimento de novos produtos é uma sequência de ligações entre as necessidades de se conhecer o mercado e as oportunidades tecnológicas que são traduzidas em informações para a produção.

Quanto ao tipo de produto mais adequado a Engenharia Simultânea, observa-se que os inovadores, apresentam maior redução no custo de criação em relação à outros tipos de produtos ao usar a Engenharia Simultânea (VALLE et al., 2009).

Quanto às ferramentas usadas, Evans (1991) demonstra que há três grandes categorias de ferramentas para a Engenharia Simultânea: as ferramentas de base tecnológica, as técnicas e a ferramenta essencial que é o trabalho em equipe. Após a apresentação dos pontos fortes e fracos associados à Engenharia Simultânea são apresentadas as ferramentas que tenham relação direta com a produtividade.

2.4.1 Pontos fortes

O desenvolvimento de um sistema para satélite espacial é dividido em subsistemas e a engenharia simultânea é uma abordagem eficiente neste caso, devido ao fato do projeto dos subsistemas ser simultâneo a iteração para elaboração do *design* destes subsistemas é mais rápida devido ao paralelismo, pois iniciam em paralelo, ao invés, de esperar o término do outro subsistema. (STEVENS, 2015)

Segundo Prassad (1996), ao implantar a Engenharia Simultânea, há uma significativa melhora da qualidade e redução no tempo de desenvolvimento, mudanças de projeto, refugos, retrabalho, defeitos, tempo de introdução do produto e frequência de falha de campo.

A Tabela 1 mostra, em percentuais, os ganhos obtidos pelas empresas com a aplicação da Engenharia Simultânea ao invés da sequencial.

Tabela 1 – Percentuais obtidos com a implantação da Engenharia Simultânea.

tempo de desenvolvimento	**30 – 50% menor**
mudanças de engenharia	60 – 95% menor
refugos e retrabalhos	**75% de redução**
defeitos	30 – 85% menor
tempo de introdução do produto	**20 – 90% menor**
frequência de falha de campo	60% menor
qualidade em geral	**100 – 600% maior**

Fonte: Mills; Bechert; Carrabine (1991); Hartley (1992); Prasad (1996)

O aumento da produtividade é um ponto forte da Engenharia Simultânea segundo Mills, Bechert e Carrabine (1991); Carter e Baker (1991); Hartley (1992); Prasad (1996); Pires e Loureiro (2010).

É observado melhora na curva de aprendizado dos estudantes no curso de programação robótica com a adoção da engenharia simultânea. A hipótese é que o aprendizado iterativo por meio da adoção da engenharia simultânea no curso é mais eficaz na construção de conhecimento, facilita a aprendizagem. (SHIN et al., 2015)

2.4.2 Pontos fracos

As restrições quanto a Engenharia Simultânea, apontadas por Rosenblatt e Watson (1991), Haddad (1996) e Sprague et al. (1991) são: a cultura organizacional é a tradicional separação entre as várias funções dentro da organização, que normalmente inviabiliza mudanças dessa natureza com as inconstâncias de propósitos. A Engenharia Simultânea age integrando as várias funções dentro de uma cultura de equipes. A falta de comunicação, que pode ser devido à distância física entre os engenheiros de desenvolvimento e os engenheiros de produção, ou à falta de experiência, principalmente dos engenheiros de desenvolvimento nas atuações da produção, ou à falta de infraestrutura. O uso intensivo de equipes e de ferramentas de interação entre projeto e processo podem amenizar este problema. Em relação a centralização de poder, há sempre o receio da perda de poder do primeiro escalão e com isso ocorre a dúvida entre aquilo que deve realmente ser feito e aquilo que os superiores instruem para ser feito. Influências externas dos clientes e fornecedores, que podem alterar consideravelmente o caminhar do projeto, principalmente se mal gerenciados. A necessidade de *software* que apoie consistentemente a colaboração e trabalho em equipe para resistência humana devido a não compreensão dos benefícios e riscos. (DEBONI et al., 1995)

Autores como Albin e Crefelt (1994); Carter e Baker (1991); Lu (1990); Pawar e Sharifi (2000); Weston (1996); Winner et al. (1988); Yan (1999); Castka e Bamber et al. (2001) apontam as principais falhas da Engenharia Simultânea que são: a falta de empenho para os problemas que surgem em função do compartilhamento dos recursos e a sua implementação necessita de importantes reformas na atual estrutura e cultura organizacional da empresa.

Muitos autores afirmam que a adoção de uma cultura para desenvolvimento de *software* aderente aos métodos ágeis requer que o grupo aceite mudanças significativas em seu processo de desenvolvimento e que isso muitas vezes exige uma mudança significativa na cultura organizacional (MCAVOY e BUTLER, 2007).

A mudança de valores que a cultura organizacional inerente aos métodos ágeis representa na organização é um processo de aprendizagem (ROBINSON e SHARP, 2003; FRASER et al., 2008). Os valores e princípios dos métodos ágeis, intrínsecos à prática da programação em pares resolvem os problemas que surgem devido ao compartilhamento de recursos. Uma oportunidade encontrada por este trabalho para a fusão com a programação em pares, é o fato dela ser adequada a

pequenas e médias empresas. Os métodos ágeis, incluindo o eXtreme Programming e a prática da programação em pares, reformam a cultura e a estrutura organizacional da empresa para colaboração, resolvendo outra falha da Engenharia Simultânea relatada por vários autores da necessidade de importantes reformas na atual estrutura e cultura organizacional da empresa.

2.4.3 Equipes de trabalho

Para que ocorra o trabalho em equipe, é necessária a realização de mudanças no relacionamento da empresa com os empregados e estes entre si próprios. A mudança organizacional ajuda a melhorar o relacionamento e deve ocorrer para que todos não se sintam inibidos. Os administradores devem estar empenhados nestas mudanças e são eles que começam o processo de implantação da Engenharia Simultânea (PITHON, 2004).

A diferença básica entre um grupo de trabalho tradicional e uma equipe de trabalho da Engenharia Simultânea está na interdependência, ou seja, uma equipe é formada por um grupo de pessoas com alto grau de interdependência entre os componentes, direcionada para a realização de uma meta ou a conclusão de uma tarefa. Uma equipe de trabalho pode ser definida como:

> Um grupo de empregados que trabalham através de uma meta comum, atuando uns sobre os outros para partilharem informações sobre os melhores procedimentos ou práticas, e tomando decisões as quais encorajem todos os membros da equipe a atuarem com todas as suas potencialidades. (Katzenbach e Smith, 1993; Mussnug e Hughey, 1997).

A formação da equipe é imperativa para a existência da Engenharia Simultânea. Uma equipe satisfeita resolve conflitos, compartilha experiências, trabalha bem em conjunto, o *stress* é baixo e o nível de confiança é alto. Quanto à eficiência dos objetivos projetados, normalmente atingem: o custo do produto estabelecido inicialmente, o orçamento de investimentos previsto para o projeto, o ciclo estabelecido e principalmente a qualidade (HAUPTMAN e HIRJI, 1996).

Desempenho é o ponto crucial para as equipes. Diversos fenômenos bastante conhecidos explicam por que as equipes apresentam bom desempenho. Elas conseguem reunir conhecimentos e experiências complementares que, por definição, excedem as de qualquer indivíduo participante da equipe. Essa mescla de conhecimento e habilidade capacita as equipes a reagir a desafios complexos, tais como inovação, qualidade e serviço ao cliente. As equipes ao desenvolverem metas e abordagens claras, elas estabelecem comunicações que dão suporte à solução de problemas e à iniciativa em tempo real. As equipes são flexíveis em resposta a

variações ocorridas em eventos e em exigências. Consequentemente, as equipes podem ajustar sua abordagem às novas informações e desafios com maior velocidade, precisão e eficácia, do que fariam indivíduos surpreendidos em meio a uma malha com maior quantidade de interligações organizacionais. Equipes oferecem uma dimensão social única, que dá realce aos aspectos econômicos e administrativos do trabalho. Equipes reais não se desenvolvem enquanto as pessoas envolvidas não trabalharem duro para superar as barreiras que se encontram no caminho em direção ao desempenho coletivo. A superação das barreiras ao desempenho é a forma de os grupos se transformarem em equipes (Katzenbach e Smith, 1993).

O trabalho em equipe é importante para a Engenharia Simultânea. O trabalho em equipe não se constitui em uma tarefa fácil de ser implementada, pois há uma grande quantidade de opiniões, informações, pessoas, etc., para serem administradas. Entretanto, é necessário que todos os elementos sejam coordenados, a fim de que os conflitos sejam diminuídos ou controlados e os canais de comunicação tornem-se mais eficientes, etc. Para o sucesso desta coordenação torna-se então imprescindível o trabalho colaborativo. (Galdámez, 2000)

2.4.4 Trabalho colaborativo

O trabalho cooperativo é aquele em que várias pessoas articulam, separadas fisicamente ou não, a realização de uma tarefa comum. (PITHON, 2004)

Cooperar é, acima de tudo, um ato social e requer, portanto, todos os tipos de interação humana, desde a fala, passando pela escrita e pelas expressões faciais. Cooperar pode ser considerado, também, um acordo em que todos se comprometem a trabalhar para atingir um objetivo comum (Borges, 1995).

A colaboração, a troca de informação, a capacidade de comunicação, o respeito às diferenças individuais e o exercício da negociação são requisitos importantes para o trabalho colaborativo. O papel da comunicação é fundamental, podendo ser realizado de várias formas, através de encontros face a face ou por meios eletrônicos. (PITHON, 2004)

Os benefícios do trabalho cooperativo podem ser medidos em termos dos objetivos da organização, os quais podem frequentemente ser generalizados por conseguir o aperfeiçoamento em curto espaço de tempo, ou melhorar a produtividade ou a qualidade. Hawryszkiewycz (1997) relaciona alguns destes benefícios: melhor uso do tempo disponível pelas pessoas através da redução da

soma do tempo gasto com a realização de tarefas rotineiras; melhor uso da informação pelo melhor aproveitamento do acesso a ela e garantir que todos compartilhem desse acesso; redução dos custos com viagens.

2.4.5 *Groupware*

O termo *Groupware* pode ser definido a partir da seguinte união de conceitos: processos e procedimentos intencionais de um **grupo** para realizar propostas específicas + ferramentas de ***softWARE*** projetadas para suportar e facilitar o trabalho em grupo. Assim, o termo *Groupware* pode ser descrito como a implementação do trabalho colaborativo em nível de *software*, com o objetivo de facilitar a comunicação colaborativa e a coordenação das ações entre as diversas pessoas a fim de promover a integração e a criatividade dentro da empresa. Esta comunicação pode ser feita através de uma rede interna (*Intranet*) ou através da *Internet* (CRUZ, 2000; HAWRYSZKIEWYCZ, 1997 e GOUVEIA, 2002).

Groupware, segundo a definição de Cruz (2000) é todo e qualquer sistema computadorizado que permite que grupos de pessoas trabalhem de forma cooperativa, a fim de atingir um objetivo comum, aumentando-lhes a produtividade.

O uso de *Groupware* deve estar respaldado pela alta direção, pois a cultura organizacional da empresa deve estar preparada para realizar as tarefas de pensar, aprender, escrever, projetar, criar, analisar, decidir, fazer *brainstorms*, dividir informação, discutir e apresentar ideias, tudo isso colaborativamente. (PITHON, 2004)

Como causas para o surgimento da tecnologia *Groupware*, Cruz (2000) aponta os fatores: *Downsizing,* Reengenharia e Programas de qualidade. *Downsizing*, redução no tamanho das estruturas organizacionais, ocorrida no final da década de 80, devido à necessidade de aumento da eficiência das empresas, para que pudessem competir num mercado globalizado. Teve reflexos diretos na área de computação. Com a substituição de *mainframes* por máquinas menores e com o surgimento de duas novas tecnologias: as redes locais e o modelo servidor, também conhecido como plataforma distribuída. Desde então, as tecnologias de rede não deixaram de crescer e consolidar-se. *Hardware* e *software*, juntos, mais próximos dos usuários, viabilizaram a ideia de trabalho produtivo em grupo. A plataforma cliente-servidor foi um fator decisivo para o surgimento e difusão da tecnologia de *groupware*. Reengenharia é um termo criado por Michael Hammer (1990) para designar o desenvolvimento de novas formas para realizar o trabalho. Para Hammer

e Champy (1994) a preocupação original de Hammer estava em fazer a reengenharia a qualquer custo, sem preocupar-se demasiadamente com a introdução de novas tecnologias da informação. Posteriormente, o significado do termo foi associado à reinvenção de processos, para aperfeiçoar o que vinha sendo feito e, desta forma, reduzir perdas. A partir do surgimento da reengenharia passou a ser incentivada a criatividade, o trabalho em grupo e o envolvimento de todos os níveis da organização no processo decisório. Programas de qualidade: devido à forte preocupação em organizar os processos de produção, a estrutura organizacional de uma empresa está intimamente ligada ao trabalho em conjunto dos membros do processo (funcionários). O *Groupware* é um dos responsáveis pela manutenção das certificações obtidas para os processos, em cada auditoria realizada.

O *Groupware* pode ser considerado um polivalente, pois engloba diversas tecnologias baseadas no mesmo princípio: pessoas trabalhando juntas para que as atividades sejam realizadas com sucesso em todas as partes do processo, independente de quem as desenvolva. (CRUZ, 2000)

Resumindo, qualquer produto que permita trabalho cooperativo com ganho de produtividade, num determinado processo, pode ser considerado membro da família *Groupware*. (PITHON, 2004)

O desenvolvimento de métodos que levem em consideração aspectos cognitivos e sociais no trabalho colaborativo passa a ser de grande importância, que objetivem a melhoria dos *Groupwares* adequando-os às características humanas. (MOECKEL e AZEVEDO, 2005)

2.4.6 Trabalho simultâneo

O paralelismo (simultaneidade) na execução das etapas de desenvolvimento de produtos ocupa uma posição de destaque no contexto da Engenharia Simultânea, uma vez que é fundamental na redução do ciclo de desenvolvimento de produtos, reduzindo desta maneira o *time-to-market* (tempo transcorrido desde a detecção da necessidade, até a introdução de um novo produto no mercado) o que constitui uma importante vantagem competitiva. Para tal é vital que haja a integração entre áreas de conhecimento, mediante a constituição de equipes multidisciplinares. A redução dos tempos de lançamento e retrabalho após o lançamento; aumenta o comprometimento dos membros da equipe com as decisões tomadas. (PIMENTEL e AUGUSTO, 2003)

A integração das informações do produto permite que todos os membros da equipe tenham acesso a informações atualizadas e comuns a todos, facilitando o processo decisório e diminuindo os riscos de decisões baseadas em informações desatualizadas. (PIMENTEL e AUGUSTO, 2003)

O paralelismo (simultaneidade) bem como a integração, entre as diversas áreas envolvidas nas etapas de desenvolvimento são algumas das diferenças em relação à engenharia sequencial.

A estratégia de simultaneidade, com o emprego de equipes multidisciplinares, fornece uma oportunidade para tratar antecipadamente, no processo de desenvolvimento, de fontes de conflitos entre agentes do desenvolvimento, que representam os pontos de vista de diferentes áreas: projeto do produto, planejamento da produção, fabricação.

Segundo Winner (2000 e 2002) o custo tende a ser reduzido sobremaneira, principalmente devido à participação do pessoal de produção nas equipes multidisciplinares, desde o início do desenvolvimento, de modo que participando no desenvolvimento, contribuem para que quando da fabricação não seja detectada a necessidade de correções no projeto (reprojetos), que ocasionam não só atrasos no lançamento de produtos, que representam elevado custo em um mercado competitivo, como também evitam os elevados custos que representam as alterações de projeto, que aumentam drasticamente à medida que o projeto se aproxima da etapa de produção, pois quanto mais próximo desta etapa, maior é a quantidade de itens integrados, e consequentemente maior é a quantidade de interfaces que necessitam de revisão em função de alterações a serem introduzidas. A participação do cliente nas equipes multidisciplinares, desde o início, é importante na medida em que permite um perfeito entendimento, por parte da equipe multidisciplinar, dos requisitos que o produto deve satisfazer. Assim como o cliente pode ser assessorado com relação a requisitos conflitantes, que possam trazer como consequência degradação na operação do produto final. Deste modo, esse entendimento provoca, desde o início, um desenvolvimento contínuo, sob uma estrutura sólida (requisitos do produto), perfeitamente compreendida e edificada por todos os membros da equipe multidisciplinar.

Na aplicação da Engenharia Simultânea é fundamental que as equipes multidisciplinares de projeto trabalhem em todos os aspectos do desenvolvimento de forma paralela e simultânea, em contraste com o processo tradicional, resultando

com isso na necessidade imperiosa de maior integração entre os dados do produto. Tais informações devem estar localizadas em uma base de dados única, ao invés de se encontrarem fragmentadas, como no caso do desenvolvimento sequencial.

Para Terwiesch, Loch e De Meyer (2002) a questão chave na coordenação de tarefas simultâneas não é a frequência da troca de informação, mas sim quais informações devem ser trocadas, em qual momento e como reagir a ela. Além disso, a troca de informações preliminares, resultada da combinação de interdependência e simultaneidade, tem sua hora apropriada para ocorrer, é proporcional aos avanços na resolução dos problemas, se a incerteza é causada por eventos externos, como esses eventos se desdobram influi no momento certo para a troca de informação.

2.4.7 Equipe multidisciplinar

Para alcançar as propostas da Engenharia Simultânea, é fundamental a formação de uma equipe multidisciplinar com pessoas de todas as áreas e especialidades envolvidas no projeto. Esta equipe pode crescer ou diminuir ao longo de sua existência, mantendo sempre um mesmo núcleo de pessoas que acompanham o desenvolvimento. A equipe deve trabalhar em sintonia, considerando todos os detalhes, para que o trabalho realizado em cada área disciplinar seja compatível com as demais e que cada área influa na outra com informações corretas e no tempo certo.

Como afirma Haddad (1996) em pesquisa realizada numa indústria automobilística americana, o uso de equipes multifuncionais proporciona responsabilidade pelo projeto do início ao fim, e assim, permite aos membros da equipe um maior poder de decisão nos vários níveis e gera um espírito coletivo de propriedade pelo projeto.

Uma característica importante desta equipe de Engenharia Simultânea é ser responsável por todo o projeto e possuir autoridade para as decisões. Esta atitude requer treinamento dos membros do time e da gerência para ser efetiva. É preciso que exista a comunicação efetiva entre os seus integrantes. Esta comunicação envolve as pessoas, a troca de dados entre os sistemas utilizados. (PITHON, 2004)

Equipes multifuncionais (também conhecidas como “força-tarefa”) que são as responsáveis pela instalação da simultaneidade dos processos. O líder da equipe, além da função de liderança (que inclui a gerência do grupo e outros aspectos da liderança do grupo), serve também como o elemento de ligação entre a diretoria e o grupo (membros). (PITHON, 2004)

3 PROGRAMAÇÃO E REVISÃO SIMULTÂNEA EM PARES

3.1 Introdução

A revisão bibliográfica indica que os pontos fortes da engenharia simultânea e revisão de código, podem ser a solução para o problema de produtividade da programação em pares e em determinadas situações o *mob programming* pode ser uma escolha vantajosa e sugere que um possível estudo terciário da programação em pares, *mob programming, code review* e engenharia simultânea, possa resultar numa nova técnica baseada na fusão entre estas técnicas correlatas e complementares entre si. Esta seção apresenta uma proposta para esta nova técnica, aqui chamada de PrsP (Programação e revisão simultânea em Pares).

3.2 Definição da Programação e revisão simultânea em Pares

A Programação e revisão simultânea em Pares é definida como: uma atividade de programação onde no começo, é feito o planejamento, a seleção dos pares, multidisciplinaridade, é projetado o emparelhamento das tarefas e com base nisso, dois programadores trabalham colaborativamente na mesma atividade, apenas no início da atividade sentados lado a lado para trocar experiências (desta forma há maior número de algoritmos e soluções) ou comunicam-se no início, se estiverem trabalhando de forma distribuída (locais diferentes), ainda nesta fase inicial, decidem como dividir a tarefa e, não precisam sentar junto o tempo todo em um único computador, ou comunicar-se o tempo todo se estiverem trabalhando de forma distribuída, somente quando necessário e útil. Sempre que possível, o trabalho deve ser executado de maneira simultânea, em computadores separados. Diferentemente da programação em pares, na PrsP, um revisa o trabalho do outro de forma simultânea e, caso for encontrado erro, o trabalho retorna para o programador corrigir. No final, unem o trabalho do par e durante o descanso, sugere-se falar ou pensar sobre a melhor maneira para a realização do trabalho, *mindset zero defect,* adoção de um processo de redução de *stress* e para resolução de conflitos.

3.3 Descrição

Ao incorporar as práticas da Engenharia Simultânea na programação em pares, deve-se notar a importância da comunicação, mesmo se a atividade for realizada em dependências físicas diferentes. Também é importante frisar que quanto mais a atividade for desenvolvida de maneira simultânea, maior será o ganho de produtividade e a comunicação e colaboração são os pilares para junção do

trabalho realizado simultaneamente em computadores diferentes com sucesso, sem comprometer com erros durante a junção do trabalho, o ganho de produtividade obtido pelo paralelismo durante sua execução.

Os desenvolvedores trabalham em pares para realizarem suas tarefas. Isso promove o trabalho coletivo, colaborativo, une a equipe, melhora a comunicação e a qualidade do código. O trabalho é desenvolvido de maneira simultânea e se houver mais de um par na equipe,as iterações são projetadas para serem simultâneas.

No projeto do emparelhamento, é definido colaborativamente a adoção ou não do *pair/mob programming* e caso for adotado, definir em quais requisitos de *software* será usada tal técnica. Também colaborativamente identifica-se as vulnerabilidades técnicas da equipe, para ajudar a direcionar um *coding dojo*, para aumentar o nível técnico da equipe, o que por sua vez, pode contribuir para aumentar a produtividade. Conforme defendem Coplien e Harrison (2005) é adotado *design* compatível com o emparelhamento de trabalhar em conjunto; desta forma podem produzir mais do que a soma dos dois individualmente. A seleção dos pares depende do projeto, da tarefa a ser realizada, da disponibilidade dos membros da equipe, da necessidade de disseminação do conhecimento, rotação dos pares e da experiência de cada um.

Programadores, se deixados à sua própria sorte, ignoram os erros mais flagrantes da sua tela, cujo os quais, qualquer outro programador enxergaria em um instante (WEINBERG, 1998). Por isso, é importante que a revisão seja realizada por outro programador, que não esteja com o olhar viciado devido a uma limitação da capacidade humana, de enxergar apenas, o que sua mente está prestando atenção.

É importante no final, que a revisão seja feita pelo par, mesmo usando dois computadores é possível beneficiar-se do mecanismo mencionado pelo Wray (2010) de avisar mais detalhes, pois ele esta baseado na característica humana de não ver o que não quer ver, então qualquer outro programador enxerga um erro, por mais flagrante que seja, de maneira mais fácil e rápida, aumentando a produtividade.

Pode-se utilizar *Groupware* para automatizar, controlar, rastrear e verificar por meio de *software* os procedimentos realizados pela equipe multidisciplinar no desenvolvimento do trabalho colaborativo e simultâneo. Pode ser vantajoso o seu uso se a equipe valorizar ou necessitar controlar ou rastrear as atividades dos pares, bem como obter dados para alguma métrica usada em algum indicador escolhido pela equipe, porém, equipes ágeis, tipicamente abrem mão do controle em prol da comunicação e confiança. Alta qualidade é priorizada pelo *mindset zero defect.*

3.4 Fases

A aplicação da PrsP é dividida em sete fases.

A primeira fase é para o planejamento. É feita a seleção dos pares, considerando a multidisciplinaridade, por exemplo, um especialista em programação com um par especialista em interação humano computador, para evitar algum problema na junção da interface com a programação. É projetado o emparelhamento das tarefas, se houver mais de um par na equipe, usar o trabalho simultâneo da engenharia simultânea para projetar iterações (*sprints)* simultâneas.

As fases dois e três, são referentes a autoridade e responsabilidade do par para a divisão do trabalho, onde os desenvolvedores buscam a melhor forma para desenvolver em paralelo/simultâneo o respectivo requisito de *software* que irão implementar em par, eles conversam na fase 2, desta maneira há um maior número de algoritmos e soluções. O par entra num acordo na fase 3 de como o trabalho será dividido, para sempre que for mais produtivo usarem dois computadores. É muito importante que esta divisão leve em consideração a junção que ocorrerá no final, para não perder o aumento da produtividade obtido com o trabalho simultâneo devido a algum problema na junção da atividade. O par tem autoridade para dividir da melhor forma o trabalho e a responsabilidade pela eficácia da divisão e união na fase final.

Na fase quatro, é realizado o trabalho de maneira simultânea, em dois computadores distintos, cada um desenvolve e testa a sua parte. O par comunica-se quando necessário, porém, sempre que possível a programação é desenvolvida de maneira paralelo/simultânea na busca pelo aumento da produtividade.

Na fase cinco, de revisão, cada desenvolvedor revisa e testa o algoritmo feito pelo par de maneira simultânea, devido a limitação humana de enxergar somente o que nota, consequentemente, qualquer outro programador enxergará um erro, por mais flagrante que seja, de maneira mais fácil e rápida, aumentando a produtividade e, isso cria a pressão do par amplamente citada na literatura, fator este que aumenta a qualidade pois outra pessoa irá verificar o trabalho, além de ser uma inspeção informal. Quando o desenvolvedor testa o seu próprio trabalho muitas vezes o seu raciocínio pode-se afirmar que fica viciado sempre nos mesmos testes, é mais difícil para quem desenvolveu o trabalho, ter um olhar de fora, pensar numa forma diferente para realizar o teste. A nova técnica além deste beneficio já amplamente conhecido de outro desenvolvedor realizar o teste, para aumentar a produtividade o

par conversou no início, então o par que irá realizar o teste, já tem uma visão do problema, o que é mais produtivo e preserva o olhar de um outro desenvolvedor efetuar o teste. O fato do par ser responsável pelo trabalho, pode aumentar o capricho do par no teste.

Durante a sexta fase, os desenvolvedores unem o código fonte, o que estimula o trabalho coletivo e colaborativo, pois para realizar a união do programa desenvolvido pelo par sem problemas, é necessário um planejamento prévio eficaz, na fase inicial. A comunicação também merece destaque nesta fase, o par deve estar alinhado em relação a como realizar a união do trabalho.

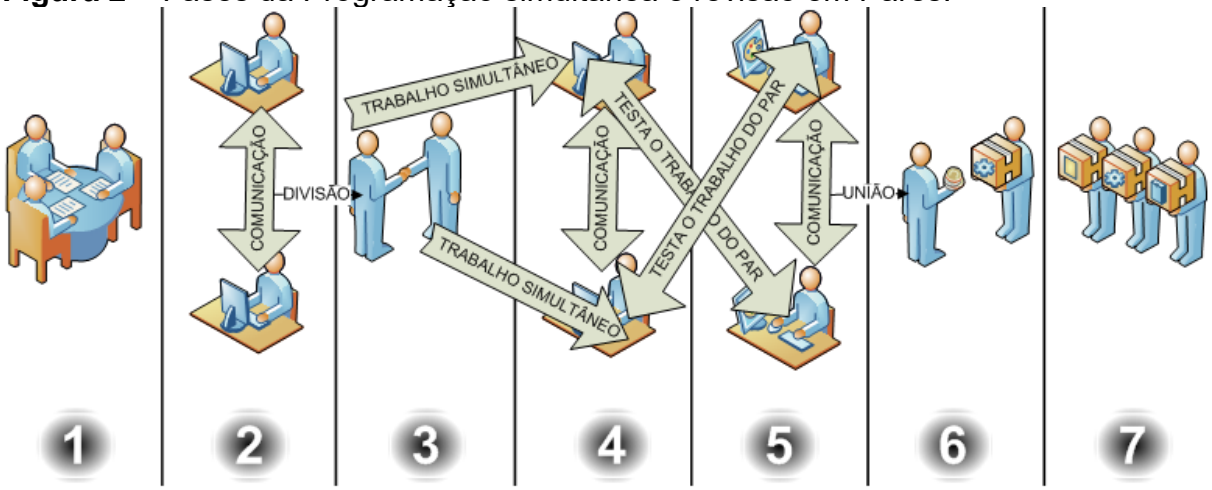
A fase 7, descanso reflexivo e prevenção/resolução de conflitos, é aberta, devido as muitas possibilidades de um descanso produtivo e maneiras de prevenção/resolução de conflitos, falar ou pensar sobre o programa/trabalho ajuda a encontrar a melhor solução, a ideia é estimular o estabelecimento de um ritmo saudável de trabalho, sem muitas horas extras por exemplo, e logo após a saída do trabalho, pensar ou falar com o colega sobre o algoritmo/trabalho. A sugestão é divertir-se, descansar, relaxar, encontrar familiares e amigos, porém, ao invés de reclamar, lamentar, falar ou pensar mal de outrem, falar ou pensar sobre algo construtivo, por exemplo, a busca por uma melhor solução para o algoritmo/trabalho, obviamente é mais produtivo e pode contribuir para um ambiente de trabalho mais harmônico e motivador, com menos conflitos, o que pode contribuir para o aumento da produtividade da equipe. Manter-se motivado é mais produtivo em relação à desmotivado. Reclamar ou lamentar-se é contraproducente, pode dificultar a percepção de soluções corretas e ao seu alcance. O respeito ao próximo e solidariedade ajudam na prevenção de conflitos. Se houver conflito, conversar para resolve-lo, colocar-se no lugar do outro é importante, bem como, manter a imparcialidade, medir cuidadosamente as palavras e se dispor a escutar.

Alguns conflitos são inevitáveis nas relações humanas. Quando as ações de alguém são controladas por outra pessoa, ocorre a possibilidade de conflito (PITHON, 2004). Trabalho em equipe produtivo com alto desempenho, requer liderança, gerenciamento congruente orientado às pessoas, motivação e resolução de conflitos entre os membros da equipe. (WEINBERG, 1997)

As fases 1,2,3,4,5 e 6 da Programação e revisão simultânea em Pares estão ilustradas na Figura 2. A fase 7, descanso reflexivo e resolução de conflitos, tem

formato aberto, devido as várias possibilidades para um descanso reflexivo/produtivo e resolução de conflitos, então está ilustrada na forma da equipe com o trabalho.

Figura 2 – Fases da Programação simultânea e revisão em Pares.



Fonte: Elaborado pelo autor

A Engenharia Simultânea tem características que contribuem ao aumento da produtividade e algumas já são intrínsecas à programação em pares. Tais características da Engenharia Simultânea que são fundidas à programação em pares por este trabalho e as que são intrínsecas estão ilustradas no Quadro 3.

Quadro 3 – Complementares ou já incorporadas à programação em pares.

Engenharia Simultânea	**Característica complementar ou já incorporada na programação em pares**
Trabalho simultâneo	Complementar, pois é difícil trabalhar de maneira simultânea usando apenas um teclado e mouse. A incorporação do trabalho simultâneo à programação em pares, ajuda a aumentar o comprometimento originário da posse do teclado
Groupware	Complementar, este trabalho sugere usar um *groupware* para apoiar o trabalho colaborativo e com isso obter um ganho de produtividade.
Equipe de trabalho	Já incorporada à programação em pares, pois o par trabalha em equipe, este trabalho apenas reorganiza o seu formato.
Trabalho colaborativo	Já incorporada à programação em pares, o trabalho colaborativo é ajustado por este trabalho para ser simultâneo
Equipe multidisciplinar	Complementar, na programação em pares são dois programadores e apenas um teclado. Para formar um par multidisciplinar este trabalho, através da nova técnica, usa dois computadores e no projeto do emparelhamento por exemplo, pode-se formar um par composto por um *designer* e um programador.

Fonte: Elaborado pelo autor.

A Engenharia Simultânea tem características que aumentam a produtividade no desenvolvimento de *software*, que justamente é o problema apresentado pela programação em pares. O Quadro 6 analisou tais características para saber se de alguma forma já fazem parte da programação em pares, se podem ser melhoradas, se é necessário adaptá-las para fundí-las ou se simplesmente são novas

características que são incorporadas à programação em pares, tais como: o trabalho simultâneo e multidisciplinar. O trabalho simultâneo, pode contribuir ao aumento da produtividade, mas é necessário adaptar a programação em pares, devido à incorporação de dois teclados e dois mouses, mas pode resolver o problema relatado na literatura sobre a falta de comprometimento quando o programador não está na posse do teclado. O trabalho em par auxiliado por *Groupware,* pode incrementar a produtividade devido ao auxílio no gerenciamento do compartilhamento dos recursos. O trabalho em equipe e colaborativo são adaptados para suportarem o trabalho em par simultâneo. A programação usando pares multidisciplinares, é incorporada à nova técnica, para melhorar a produtividade, baseando-se no aumento da produtividade que as equipes multidisciplinares geram à Engenharia Simultânea.

Este trabalho descreve os pontos fortes e fracos da programação em pares, *mob programming,* revisão de código e engenharia simultânea. Uma fusão soa natural ao analisar-se o fato do ponto forte de uma ser o ponto fraco da outra e vice versa. O ponto fraco da programação em pares constatado por Smith et al. (2002); Parrish et al. (2004); Arisholm et al. (2007) e Dyba et al. (2007) é a produtividade, aumento do esforço de trabalho por pessoa/hora, ao considerar-se programadores experientes ou tarefas simples. A produtividade é justamente, o ponto forte da Engenharia Simultânea, pois segundo Mills, Bechert e Carrabine (1991); Carter e Baker (1991); Hartley (1992) e Prasad (1996); Pires e Loureiro (2010) ela aumenta a produtividade e o autor observou que nestas publicações os profissionais ou programadores eram experientes.

A mudança de valores, que a cultura organizacional inerente aos métodos ágeis representa na organização, observada por Robinson e Sharp (2003); Mcavoy e Butler (2007) e Fraser et al. (2008) pode auxiliar na melhora dos pontos fracos da Engenharia Simultânea apontados por Watson (1991), Haddad (1996) e Sprague et al. (1991) acerca da necessidade de adaptação da cultura organizacional, integração das funções em uma cultura de equipes. Os valores e princípios dos métodos ágeis, intrínsecos à prática da programação em pares resolvem o problema da falta de empenho oriundos dos problemas que surgem devido ao compartilhamento de recursos na Engenharia Simultânea. Uma oportunidade encontrada por este trabalho para a fusão com a programação em pares, é o fato dela ser adequada a pequenas e médias empresas, que é justamente outro ponto fraco da Engenharia Simultânea.

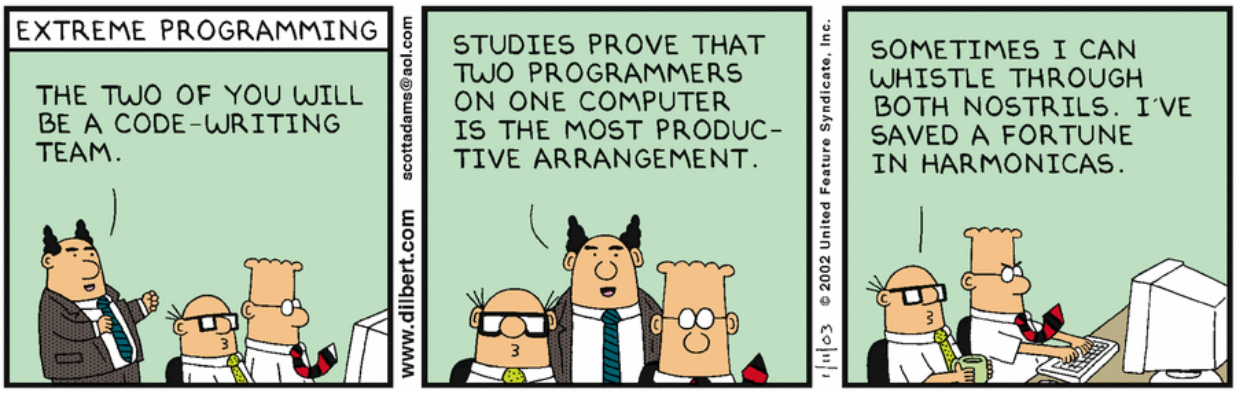
O trabalho simultâneo, multidisciplinar e auxiliado por *groupware* da Engenharia Simultânea é fundido à programação em pares, o trabalho em equipe e colaborativo são mantidos e ajustados ao formato da nova técnica, os mecanismos para aumentar a produtividade: avisar mais detalhes, conversar, lutar contra práticas ruins de programação e compartilhar e reconhecer competências citados por Wray (2010) são incorporados à nova técnica, bem como a rotação dos pares na equipe observada por Swamidurai e Umphress (2014) que gera aumento da produtividade.

Na tentativa de resolver o problema de comprometimento ocasionado pela falta da posse do teclado citado por Vanhanen e Korpi (2007); Williams e Kessler (2003), é acatada a sugestão para possível resolução deste problema, de usar dois teclados e dois mouses, fornecida por Chong e Hurlbutt (2007); Freudenberg, Romero e Boulay (2007) e Höfer (2008). A nova técnica PrsP incorpora à prática de revisões de código em pares e adere ao *coding dojo,* para disseminar e aumentar conhecimento técnico da equipe. Para a disseminação do conhecimento operacional e para os códigos críticos e complexos pode-se usar o *mob programming*.

3.5 Dois teclados ao invés de um, ajuda a manter o comprometimento do par

O brilhante cartunista Scott Adams (2003) ilustra na Figura 10, o problema de comprometimento ocasionado pela falta da posse do teclado citado por Vanhanen e Korpi (2007); Williams e Kessler (2003). Na Figura 3, o Dilbert (está usando óculos redondos) esta na posse do teclado. Chong e Hurlbutt (2007); Freudenberg, Romero e Boulay (2007); Höfer (2008), sugerem usar dois teclados e dois mouses como uma forma para manter os dois programadores do par comprometidos. O Dilbert, provavelmente também apoiaria a ideia de dois mouses e teclados.

Figura 3 – Maior comprometimento de quem esta na posse do teclado.



Fonte: DILBERT © 2003 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

3.6 Sentar junto apenas para conversar ao invés de durante todo o expediente, pode ajudar a aumentar a produtividade

Para Terwiesch, Loch e De Meyer (2002) a questão chave na coordenação de tarefas simultâneas não é a frequência da troca de informação, mas sim quais informações devem ser trocadas, em qual momento e como reagir a ela. Este trabalho de engenharia simultânea é um dos argumentos da PrsP, para o par não sentar junto durante todo o expediente, somente para conversar, quando for preciso e útil.

Há pessoas que não gostam durante o expediente, especialmente em sua mesa de trabalho de cheiro de comida ou barulho gerado pelo par. A figura 4 mostra que o personagem Wally com sua caneca de café e seu histórico de baixo comprometimento não gostou da programação em pares do XP devido ao fato de ninguém querer ser seu par.

Figura 4 – Duas pessoas e apenas um computador durante todo expediente.

Fonte: DILBERT © 2003 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

A nova técnica proposta usa dois teclados e dois mouses para o desenvolvimento do trabalho, desta maneira é mais produtivo, pois o trabalho é executado em paralelo/simultâneo e resolve eventuais problemas causados por ficar quarenta horas sentado ao lado de outra pessoa.

Senta-se junto quando é necessário ao invés de durante todo o horário de expediente. A definição abaixo incorpora o trabalho simultâneo, usando dois computadores, conseguintemente dois mouses e dois teclados.

3.7 Fatores de produtividade

A nova técnica PrsP, afeta positivamente os fatores listados que causam problema de produtividade no desenvolvimento de *software*.

O Quadro 4 explica como a fusão entre programação em pares, *mob programming, code review* e Engenharia Simultânea atacam em conjunto o problema da produtividade no desenvolvimento de *software*, incluindo os fatores humanos, ainda muitas vezes negligenciados quando o assunto é produtividade no desenvolvimento de *software*.

Trendowicz e Münch (2009), fizeram uma revisão sistemática dos fatores que influenciam a produtividade no desenvolvimento de *software*, o principal resultado é que o sucesso dos projetos de *software,* ainda depende de seres humanos, o segundo fator mais comumente considerado são ferramenta e método. No entanto, mesmo a melhor ferramenta ou método não são balas de prata e não substituem pessoas altamente qualificadas e coordenação eficaz do trabalho. Ferramentas e métodos devem, portanto, serem considerados amplificadores do impacto positivo de equipes altamente qualificadas e bem coordenadas sobre a produtividade no desenvolvimento de *software.*

A redução de esforço pode proporcionar um aumento gradativo na produtividade ou pelo menos mantê-la estável durante o ciclo de vida de desenvolvimento (BOEHM et al., 2009).

A realidade mostra que ainda existe uma lacuna entre a demanda de mercado e a produtividade do desenvolvimento de *software* das organizações (SANTOS; MOURA, 2009).

Segundo Sampaio et al. (2010); Andrade (2012); Kon et al. (2013) e Melo (2015) vários fatores contribuem para esse cenário, como tamanho da equipe, estabilidade dos requisitos, qualidade da gestão, habilidades e experiência da equipe, reuso de *software*, ferramentas de desenvolvimento, participação e experiência do cliente, motivação da equipe, tamanho e complexidade do *software.*

Quadro 4 – Fatores de produtividade afetados pela nova técnica PrsP.

Fatores de produtividade no desenvolvimento	**Maneira como a PrsP afeta os fatores de produtividade**
Reuso de *software*	Reuso de *Software* facilitado pela disseminação do conhecimento e a rotação dos pares
Tamanho e complexidade do *software* (grandes e complexos)	Estimula a comunicação e é auxiliada por *Groupware* e simultaneidade, reduzindo o esforço no desenvolvimento

Continua na próxima página

Fatores de produtividade no desenvolvimento	Maneira como a PrsP afeta os fatores de produtividade
Motivação da equipe	Trabalho em pares estimula companheirismo, melhora a motivação, satisfação e pode reduzir a rotatividade de pessoas na equipe. Trabalho em equipe e a posse do teclado estimulam o comprometimento. Trabalho simultâneo reduz a morosidade.
Qualidade da gestão	Gestão realizada pela própria equipe, comunicação e *Groupware* auxiliam a gestão e satisfação da equipe
Habilidades e experiência da equipe	Rotação dos Pares (dissemina conhecimento). Equipes multidisciplinares. Resolução de conflitos pode ajudar a amplificar a disseminação de conhecimento por meio de um ambiente de trabalho harmônico e favorável ao diálogo.
Tamanho da equipe (equipes grandes)	Apoio de software Groupware para facilitar o trabalho colaborativo. Trabalho em equipe e colaborativo para reduzir conflitos, facilitando a gestão de equipes grandes.
Estabilidade dos requisitos	A participação do cliente nas equipes multidisciplinares, desde o início, é uma importante medida que permite um perfeito entendimento, por parte da equipe multidisciplinar, dos requisitos que o produto deve satisfazer, bem como o cliente pode ser assessorado com relação a requisitos conflitantes.
Participação e experiência do cliente	Equipe multidisciplinar e trabalho colaborativo com apoio de Groupware. *Mob Programming,* pode facilitar o engajamento de outras áreas no trabalho do grupo.
Comunicação	Equipe multidisciplinar, planejamento colaborativo e trabalho baseado em pares com comunicação constante.
Práticas modernas de programação	Na Fase 1 da PrsP, decide-se de forma colaborativa a necessidade da realização de *coding dojo* ou *mob programming* para disseminar conhecimento. O trabalho em pares inerente à técnica também pode ajudar a contribuir para o aumento do nível técnico da equipe.
*Turn ove*r	Fase 7 da PrsP, descanso reflexivo e prevensão/resolução de conflitos, trabalho em pares, equipes, *mob programming*, podem aumentar a satisfação, motivação, comunicação, colaboração e confiança, possivelmente reduzindo o *turn over.*
Gerenciamento de conflitos	Fase 7 da PrsP, descanso reflexivo e prevensão/resolução de conflitos, atua na prevenção e estabelecimento de um ambiente de trabalho harmonioso e um protocolo de resolução se tiver conflito.

Fonte: Elaborado pelo autor.

Além das explicações do quadro 4, o principal artifício da PrsP para aumentar a produtividade, é o trabalho simultâneo, que ocorre usando dois computadores, portanto dois teclados e dois mouses, diferenciando-se da programação em pares que usa apenas um computador e conseguintemente apenas um mouse e teclado. As iterações simultâneas, também podem contribuir para aumentar a produtividade.

3.8– Conclusão

Foi apresentada a nova técnica Programação e revisão simultânea em Pares, foram detalhadas as suas fases e definição. Foram apresentados os argumentos que embasam/justificam a fusão proposta com base na revisão da literatura.

4 EXEMPLOS DE APLICAÇÃO DA NOVA TÉCNICA

Nesta seção são apresentados oito exemplos de aplicação e os contextos nos quais a nova técnica proposta é aplicada, com descrição e demais detalhes dos projetos, complexidade da tarefa e experiência do programador. Para manter o sigilo são omitidos os nomes das organizações nas quais a pesquisa foi realizada e das pessoas envolvidas. O pesquisador responsável por este trabalho foi consultor das organizações nas quais este trabalho foi executado. Sua atuação no trabalho foi a de líder técnico e desenvolvedor de algumas das equipes de desenvolvimento de *software* e, o autor realizou um *workshop* sobre o funcionamento da nova técnica proposta, apresentou a forma para a coleta dos dados. Nos projetos em que o autor é um dos desenvolvedores, ele está devidamente identificado. A tabela 2 descreve os seguimentos de atuação das organizações que aplicam a PrsP, a cidade onde está situada e a cultura/método de desenvolvimento usado pela organização.

Tabela 02 - Descrição das organizações que aplicaram a nova técnica proposta

Nome da organização	**Cidade**	**Segmento de atuação**	**Cultura e método de desenvolvimento de *software***
Cliente 1	São Paulo	Tecnologia de Informação / Desenvolvimento de *Software*	Própria, método híbrido do autor e alguns projetos externos usam os métodos de desenvolvimento de *software* dos clientes contratantes.
Cliente 2	São Paulo	Instituição financeira	Formal com auditoria, baseado em planejamento.
Cliente 3	Lisboa	Tecnologia da Informação	Híbrido de ágil com planejamento, diversas auditorias e controle rígido dos projetos em relação a custo e prazo devido à lei estadunidense Sarbanes-Oxley (SOX).
Cliente 4	São Paulo	Instituição de Ensino	A PrsP é aplicada em sala de aula de cursos de programação, métodos de desenvolvimento e engenharia de *sofware.*

Fonte: Elaborado pelo autor

Para a classificação do nível de complexidade é usada uma fórmula do autor, é feita uma correlação do número obtido com a complexidade que é classificada em: baixa, média e alta. É uma aplicação das métricas bem estabelecidas e válidas, que para computar a complexidade, vai além da Complexidade Ciclomática, é levado em consideração a coesão, quanto maior for a falta de coesão (*Lack of Cohesion of Methods* ou *LCOM*), maior será o desconto na complexidade. São somados os valores DIT (*Depth of Inheritance Tree*), NOC (Number of Children), AC (Afferent Coupling) e EC (Efferent Coupling). A fórmula usada para medir a complexidade é:

$$Complexidade = Complexidade\ Ciclomática - \text{LCOM} + \text{DIT} + \text{NOC} + \text{AC} + \text{EC}$$

Padhy et al. (2015) realizaram uma revisão sistemática sobre métricas orientadas a objetos e concluíram que complexidade ciclomática, DIT, NOC, AC, EC e LCOM são bem estabelecidas e válidas. O autor deste trabalho no primeiro exemplo de aplicação, identifica os mesmos tipos de complexidade que Antinyan et al. (2015): interna e externa. A interna é a complexidade da própria função/método e a externa é sua relação de dependência com outras funções/métodos.

Antinyan et al. (2015), realizaram uma pesquisa ação na Ericsson e AB Volvo com a participação de pesquisadores das universidades de Gothenburg e Skövde, para identificar os aspectos da complexidade do código e usam as mesmas métricas usadas pelo autor em seu método de avaliação da complexidade do código, porém, seu objetivo é identificar a complexidade de código para refatorar, enquanto o objetivo do autor deste trabalho é levar em consideração a complexidade ao medir a produtividade no desenvolvimento e qualidade do produto de *software*, por isso, a fórmula do autor para complexidade, não exclui do cálculo, quando o método apresenta um número baixo de complexidade, como fazem na Ericsson e AB Volvo.

Mandhan et al. (2015) observaram significativo nível de acerto na predição da densidade de defeitos com as métricas complexidade ciclomática,DIT, NOC, AC, EC e LCOM que fazem parte do *NASA KC1 PROMISE Repository of empirical software* (NASA, 2015). Para Nicolaescu et al. (2015) os pesquisadores devem focar agora na aplicação das métricas bem estabelecidas e válidas para os mais variados propósitos. A Tabela 03 exibe a classificação da complexidade dos exemplos, bem como o número total de linhas de código e a quantidade de programadores dos projetos, bem como se a atuação é exclusiva, em tempo integral para o projeto.

Tabela 03 - Complexidade, número total de linhas de código e quantidade de programadores dos exemplos de aplicação que usaram a nova técnica PrsP

Nome da organização	Exemplo de aplicação – Nome do projeto	Comple-xidade	Quantidade de desenvol-vedores	Equipe exclusiva para o projeto?	Número total de linhas de código
Cliente 1	1-Atividades esportivas	Média	4	Não, meio período	± 25 mil
Cliente 1	2-*Homebroker*	Alta	12	Sim	± 4 milhões
Cliente 1	3-*E-commerce*	Média	4	Não	± 100 mil
Cliente 2	4-Facebroker / Farejador	Alta	4	Sim	± 600 mil
Cliente 1	5-*DMA*	Alta	8	Sim	± 1,4 milhões

Continua na próxima página

Nome da organização	Exemplo de aplicação – Nome do projeto	Comple-xidade	Quantidade de desenvol-vedores	Equipe exclusiva para o projeto?	Número total de linhas de código
Cliente 3	6-Seguradoras	Alta	12	Sim	± 10 milhões
Cliente 1	7-Gerenciador de contratos	Média	4	Sim	± 70 mil
Cliente 4	8-Exercício de Programação	Baixa	16	Sim, alunos	± 200

Fonte: Elaborado pelo autor

A experiência do desenvolvedor é classificada conforme o registro de contrato profissional de trabalho e questionário de aderência à programação em pares. É adaptado e usado o questionário utilizado no trabalho de Sato (2007), construído com base no trabalho de Krebs (2002) e está disponível online (HEREZ, 2015).

Para coletar de forma automática as métricas de código os 7 primeiros exemplos de aplicação aderentes a plataforma tecnológica Microsoft .NET, alguns dos exemplos de aplicação, usaram a ferramenta *NDepend*, que calcula acoplamento entre as classes do sistema, e *NCover* que calcula cobertura de código. Para verificar o padrão de código é usado o *FxCop*. Os exemplos de aplicação que usam as versões mais recentes e completas do *Visual Studio*®, é usado o *Visual Studio® Code Metrics Power Tool*, que calcula a complexidade ciclomática, linhas de código por método, entre outras medidas e métricas adotadas por este trabalho. Apenas o oitavo exemplo de aplicação, um exercício de programação em sala de aula, os alunos usam como ferramenta de desenvolvimento o *Eclipse* (2015) com o *plug-in Eclipse Metrics* (2015).

4.1 Medidas, métricas, indicadores e ferramentas aplicáveis à PrsP

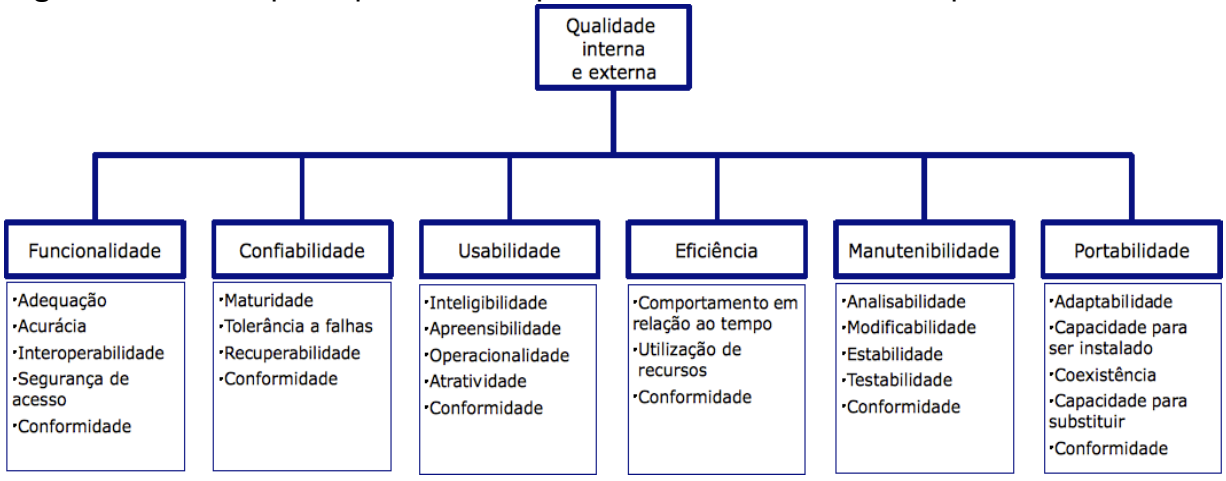
Adicionar linhas de código indica apenas que o tamanho do *software* aumentou, não aumenta o conhecimento de domínio do problema a ser resolvido, *guidelines* do projeto, conhecimento de programação ou qualquer outro aspecto do desenvolvimento de *software* (PUTNAM, 2015). Segundo o fundador da Microsoft, Bill Gates, medir a produtividade de *software* pelo número de linhas de código é como medir o progresso de um avião pelo seu peso (GATES, 2014).

A métrica proposta pelo autor para mensurar a produtividade é ligada à qualidade do produto de *software,* pois os melhores algoritmos são os mais simples que resolvem o problema (provavelmente tem menos linhas de código) e uma métrica que use apenas as medidas de contagem de linhas de código por

desenvolvedor por mês, não funciona para medir a produtividade, uma quantidade maior linhas de código ou pontos por função não significa que o desenvolvedor é mais produtivo. Não existe um padrão universal para representar linhas de código, pois as linguagens podem variar, assim como as regras para cálculo de linhas de código. Portanto, uma medida pode ser baseada em um padrão local ou universal, mas o padrão precisa ser bem definido. Neste trabalho, para melhorar a precisão desta medida, em todos os exemplos de aplicação é definido um padrão de codificação e por meio das ferramentas de análise de código, é verificada a aderência ao padrão de codificação. Em relação a métrica referente ao número de defeitos encontrados após a implantação, as medidas que compõem essa métrica são o número de defeitos e a fase (ou data) onde o defeito foi identificado.

O aumento substancial no número de defeitos encontrados na última versão é um indicador de que a qualidade do *software* piorou, porém, há outros aspectos da qualidade do produto de *software,* além da funcionalidade e do número de defeitos, para melhorar a precisão ao mensurar a qualidade do produto de *software,* é necessário levar em consideração também a manutenibilidade, confiabilidade, usabilidade, eficiência e portabilidade. Conforme padronizado na norma ISO/IEC 9126, que é uma norma ISO para qualidade de produto de *software*, que se enquadra no modelo de qualidade das normas da família 9000. A norma brasileira correspondente é a NBR ISO/IEC 9126. Esta norma refere-se a qualidade do produto de *software*, propondo Atributos de Qualidade, distribuídos em seis características principais, com cada uma delas divididas em sub características, conforme é ilustrado na Figura 5.

Figura 5 - Modelo para qualidade do produto de *software*, está na parte 1 da norma.



Fonte: NBR ISO/IEC 9126 ISO 9126, 2001

Dyba e Dingsøyr (2008) fizeram uma revisão sistemática de estudos empíricos de desenvolvimento de *software* ágil, e no que diz respeito a produtividade de equipes ágeis versus tradicionais, três dos quatro estudos comparativos usaram LOC por hora e nenhum dos estudos tinha uma estratégia apropriada para garantir uma comparação imparcial e não tendenciosa.

Toda a coleta dos dados para as métricas, neste trabalho é de forma automatizada por meio de ferramentas disponíveis no mercado. Análise estática, significa analisar o código-fonte sem executá-lo e este tipo de análise pode ser realizada por meio de ferramentas (como o *Visual Studio® 2013 Code Analysis Tool*) ou manualmente através da opção *Code Review* que é suportada no Visual Studio® 2012 e 2013. A análise de código estática ajuda os desenvolvedores a identificarem problemas com o código relacionados a *design patterns*, interoperabilidade, desempenho, segurança e várias outras categorias de potenciais problemas de acordo com as regras da Microsoft® para as melhores práticas em escrever código.

Executar uma ferramenta de análise de código em intervalos regulares durante o processo de desenvolvimento, pode melhorar a qualidade do software, examinar o código para um conjunto de defeitos e violações comuns do padrão de codificação é sempre uma boa prática de programação. Além disso, a análise de código, também pode encontrar defeitos no código que são difíceis de descobrir, através de testes que permitem atingir um alto nível de qualidade para a aplicação, ainda durante a fase de desenvolvimento. A análise de código é usada por este trabalho para ajudar na avaliação e garantia da qualidade do algoritmo, para melhorar a precisão da medida da produtividade por LOC por desenvolvedor, passa-se a considerar a qualidade. O *Team Foundation Server®,* permite a execução da análise do código no momento do *check-in* do código fonte no seu sistema de repositório de dados (controle de versão). Possibilita a criação de políticas de *check-in* contendo regras que obrigam os desenvolvedores a seguirem, previne-se código fora do padrão.

É usado o StyleCop (2015) que é uma ferramenta que analisa código fonte, não compilado, ótimo para checar endentação, comentários, composição do arquivo, padrão de nomenclatura. Também é usada a ferramenta FxCop, que é outra ferramenta de análise de código, que verifica *assemblies* .NET com código gerenciado, verifica a conformidade com as diretrizes de *design* para .NET referenciados previamente, através de configurações. Prevê regras de análise de

código e permite que sejam personalizadas as regras para o projeto. Supressão de avisos também é permitida. O autor sugere e adota em seus projetos, o uso de uma das abordagens híbrida ou não (depende o caso e esta discussão é longa e extrapola o objetivo deste trabalho, então são apenas mencionados os métodos): SAAM, ATAM, CBAM, ou o *framework* de Supakkul e Chung (2004) para tratar os requisitos não funcionais de forma prática através de um *software* de código aberto bem interessante para levar em consideração e priorizar os requisitos não funcionais, baseando-se nesta priorização dos requisitos não funcionais e usando como referência a norma ISO 9126, é configurado o FxCop, e posteriormente computado o desvio do padrão. As métricas como TLOC, complexidade ciclomática, LCOM, DIT, NOC, AC e EC são coletadas diretamente do código fonte dos exemplos por meio das ferramentas *Visual Studio® Code Metrics Power Tool, NDepends, NCover, Plug-in do Eclipse Metrics* (2015). Os arquivos são obtidos do repositório de código, nas revisões correspondentes ao final de cada iteração.

Repositório de dados (*SourceSafe®, Team Foundation Server®, Subversion* (2015) ou *Git* (2015): métricas como o número de *commits* e número de linhas código alteradas por *commit* são obtidas diretamente do repositório, através de um *script* em alguns casos, que filtra os dados necessários, analisando o histórico, *logs* e diferenças entre versões atualizadas no repositório. Essas métricas são coletadas ao final de cada iteração.

Degradação da Qualidade: para ser mais produtivo é necessário no mínimo preservar a qualidade equivalente ao objeto de comparação. Defeitos, linhas de código fora do padrão, estão diretamente ligados à qualidade do código. Esta métrica proposta pelo autor para mensurar a degradação da qualidade do código, leva em consideração os defeitos, pois podem custar vidas, ocasionar perdas e danos graves à imagem corporativa, materiais, financeiros, entre outros, por isso, é multiplicado por dois, para que os defeitos pesem com mais rigor. Na literatura da programação em pares e das métricas de *software* a qualidade é tipicamente medida apenas pelos defeitos por KLOC, este trabalho vai além dos defeitos, considera o desvio do padrão de codificação, pois código ruim, fora do padrão, degrada a qualidade. A fórmula usada para computá-la é:

$$Degradação\ da\ Qualidade = Desvio\ do\ Padrão + Densidade\ de\ Defeito * 2$$

Desvio do Padrão: representa o percentual total de linhas de código não aderentes ao padrão de codificação. É a avaliação da conformidade do código escrito com os padrões estabelecidos pelos órgãos/fabricantes competentes (ISO/OSI, W3C, PHP.NET, Microsoft .NET, Oracle Java design patterns, etc). Para Staa (2000) deve-se verificar: sintaxe, retidão e perícia ao escrever o código do *software*; perfeita utilização da linguagem de programação; utilização de comentários; identificação e cabeçalhos; otimização exigida para um código de qualidade. Conforme a priorização feita dos atributos de qualidade, é usado como referência a norma ISO/IEC 9126 na calibração das ferramentas de análise de código. Deve-se buscar nesta métrica proposta pelo autor o valor zero, pois as ferramentas ajudam durante o desenvolvimento na prevenção de código fora do padrão, porém, se não for possível, a fórmula para computar o desvio do padrão é:

$$Desvio\ do\ Padrão = \frac{TLOCOOSC}{TLOC}$$ Onde,

TLOC = número total de linhas de código de produção.

TLOCOOSC = número total de linhas de código fora do padrão de codificação.

Densidade dos defeitos: é computado pelo número de defeitos de uma unidade. Este trabalho usa para a coleta automatizada dos defeitos usa a aplicação de código aberto *Bugnet* (exemplos de aplicação com *Visual Studio*, com exceção do primeiro*)* ou *Mantis Bug Tracker* (primeiro exemplo de aplicação*)*, para prevenção contra tendenciosidade e/ou classificação errônea por parte do usuário e atendente, a classificação deve ser auditada colaborativamente por dois profissionais, um da área de negócio e outro da área de desenvolvimento de *software*. O método híbrido adotado em alguns exemplos de aplicação deste trabalho, usa uma prática japonesa que faz parte desde a primeira versão do *Microsoft® Solutions Framework,* chamada de *mindset zero defect*, nesta prática deve-se buscar a excelência no desenvolvimento, sem cometer nenhum erro.

Número de Pontos (*story-points*) ou “horas ideais”: representa o número total de pontos implementados e aceitos pelo cliente. Beck e Fowler (2001) sugerem a utilização de “horas ideais” nas estimativas e controle da iteração, mas a unidade de medida efetivamente utilizada não importa tanto, contanto que seja usada consistentemente durante o projeto. Pode ser usado para a complexidade da tarefa, ao invés de considerar-se o esforço, é levado em consideração a complexidade.

Produtividade direta: é necessário desconsiderar códigos fora do padrão e com defeitos ao computar a produtividade da equipe. Linhas de código atreladas a um padrão de codificação, não capturam a complexidade do código (mil linhas de código simples exige uma quantidade de tempo para pensar diferente de mil linhas de código complexo), que deve ser levada em consideração, pois influi ao medir-se a produtividade. Esta métrica proposta pelo autor tem lastro direto com o código fonte e binário gerado pelo programa. Para computá-la, multiplica-se TLOC pela Complexidade descontando-se a Degradação da Qualidade:

$$Produtividade\ Direta = (TLOC \times Complexidade) - ((TLOC\ X\ Complexidade) \times Degradação\ da\ Qualidade)$$

4.2 Primeiro exemplo, atividades esportivas

Primeiro o autor realiza em janeiro de 2008 um *workshop* explicando sua ideia para sua menor equipe, escolhida devido ao fato do *software* para atividades esportivas ter um risco e complexidade significativamente menor que o principal projeto da empresa, um *homebroker* (*software* para operar no mercado de capitais).

Colaborativamente, são escolhidos os requisitos a serem implementados usando a PrsP e comparados à programação em pares.

O software para atividades esportivas, teve sua primeira versão desenvolvida pelo autor em 1995, originalmente desenvolvido em Microsoft Visual Basic® 3, atualizado para Microsoft Visual Basic® 6 em 1998 e assim permaneceu por 10 anos sofrendo evoluções funcionais, o autor programou sozinho este projeto até 2005, onde passou a contar com mais 3 programadores experientes neste projeto, porém, em tempo parcial, com a priorização das atividades feita pela necessidade dos clientes da empresa do autor.

O conhecimento dos requisitos do *software*, experiência como programador, instrutor de treinamentos de programação e conhecimento da experiência dos programadores envolvidos, pois são ex-alunos do autor (treinamento para as certificações de Microsoft Visual Basic® e C# .NET), embasaram o autor a pensar que este seria o projeto ideal para realizar este primeiro exemplo como um teste para saber se a PrsP era factível e averiguar com maior profundidade se a PrsP é mais produtiva e produz código com qualidade em relação à programação em pares.

Os requisitos do *software* a serem implementado usando a PrsP são referentes a migração do aplicativo esportivo de Microsoft Visual Basic® 6 para C#.NET.

O repositório para código fonte é o *Microsoft® SourceSafe.* A coleta dos defeitos é feita de maneira automatizada através do *Mantis Bug Tracker.*

Para assegurar que o nível dos programadores é equivalente, o autor conduz um *coding dojo* no formato de *mob programming,* onde o autor de posse do teclado faz uma apresentação de todos os componentes e partes mais complexas, onde são eliminadas as dúvidas técnicas, antes do início da programação.

São atribuídos de forma colaborativa, *story points* aos requisitos e divididos em quatro partes de mesmo valor. Um par programa um quarto com a programação em pares enquanto o outro par programa o seu um quarto usando a PrsP. Depois é feita a inversão, o par que estava usando a programação em pares passa a usar a PrsP e vice-versa. Os quatro programadores sempre iniciaram na mesma hora e minuto, para facilitar a contagem da carga horária. A migração durou 4 semanas, ambas as duplas programaram 2 semanas usando a PrsP e 2 semanas usando a programação em pares. A contagem das linhas de código é feita no *Microsoft SourceSafe®.*

Os resultados são expressivos, a média obtida pela equipe neste exemplo de aplicação ao usar a PrsP é de 170 linhas de código por hora por desenvolvedor e ao usar a programação em pares é de 68 linhas de código por hora por desenvolvedor.

Observa-se neste exemplo de aplicação, uma significativa redução do esforço e duração no calendário de desenvolvimento, quando usa-se a nova técnica PrsP, a redução é de 60% em relação à programação em pares e com qualidade equivalente. A surpresa para o autor são esses 10% acima da metade, pois os programadores são experientes e capazes, a tarefa era clara, não havia incerteza, tratava-se apenas de uma migração, é normal pensar em 40% de ganho neste exemplo, pois é normal pensar em dois teclados sendo digitados por programadores equivalentes produzindo o dobro em relação à um, também é razoável, pensar num desconto adicional de 10% para os programadores dividirem o trabalho no começo e juntarem no final.

O ganho obtido por meio da sinergia do trabalho em equipe composta pela dupla simultânea é significativo, é justo pensar antes da análise deste exemplo de aplicação que a perda de dividir e juntar o trabalho seria equivalente a sinergia obtida pela dupla o que resultaria numa redução de 50% do esforço, porém, observa-se neste exemplo uma redução de 10% maior que a metade do esforço.

Este exemplo durante os 727 dias em que foi usado por 18 horas diárias não apresentou nenhum defeito, apenas solicitações de novos treinamentos, devido a entrada de novos funcionários.

Neste primeiro exemplo, o autor, começou a buscar uma forma mais precisa que a adotada por outros trabalhos da literatura da programação em pares para mensurar a qualidade, não apenas por meio da contagem do número de defeitos e produtividade apenas por LOC ou pontos por função por programador, mas levar em consideração a complexidade do código, tanto interna de uma função, quanto suas dependências externas, bem como usar a norma ISO 9126, como referência para uma avaliação acurada da qualidade do produto de *software*.

O código é inspecionado manualmente e está aderente ao padrão de codificação e sem duplicidade.

Um resultado inesperado é a opinião dos desenvolvedores a respeito da preferência pela PrsP em relação à programação em pares, devido ao aumento da satisfação, motivação, colaboração, comunicação e confiança entre os membros da equipe. A coleta é realizada por meio de um calendário *niko-niko (*NIKO-NIKO, 2015) anotado semanalmente por cada desenvolvedor num *flip-chart*. Uma possível explicação para a preferência é o fato de não precisar sentar lado a lado durante todo expediente e sempre estar na posse do teclado.

4.3 Segundo exemplo de aplicação, *homebroker*

No final de 2008, é usada a PrsP em metade do projeto de um *homebroker* (*software* para operar no mercado de capitais), a outra metade do projeto utiliza a programação em pares. É adotada como estratégia implementar os *Sprints* do *Scrum* (SCHWABER e BEEDLE, 2001), bem como, o desenvolvimento iterativo e incremental. A primeira *sprint,* o autor destina à arquitetura de *software* do projeto.

As *sprints* são baseadas em entregas funcionais, de duração variável (com duração máxima de 14 semanas), ao invés de *timebox* (duração fixa de 2 semanas).

O método usado pelo autor neste projeto é híbrido e também é baseado no *eXtreme Programming* (BECK et al., 2004) para que o desenvolvimento seja feito usando programação em pares como mencionado no trabalho, os testes são automatizados, neste método híbrido também é englobado o *Test Driven Development* (TDD). Seguindo a técnica, construímos testes de acordo com a real necessidades dos requisitos funcionais, e direcionamos o desenvolvimento das

funcionalidades a partir destes testes garantindo que estas atendam aos requisitos por consequência, desta forma é estimulada a agilidade (BECK, 2003). Todavia, o método híbrido usado neste projeto, também é fortemente baseado em *Rational Unified Process, Open Unified Process,* Engenharia Simultânea de *Software* e *Microsoft Solutions Framework,* devido a necessidade de ser um projeto bem documentado principalmente em sua arquitetura de *software* para permitir uma auditoria fácil por parte dos auditores independentes da CVM (Comissão de Valores Mobiliários) e do PQO (Programa de Qualificação Operacional) da bolsa. O autor para facilitar estas auditorias, fez uma alteração na matriz de requisitos de Firesmith (2004) acrescentando um campo para facilitar a rastreabilidade da documentação da arquitetura aos requisitos, inclusive ao código.

Neste exemplo de aplicação é mostrado apenas o emparelhamento das atividades, a primeira *sprint* referente a arquitetura de *software,* bem como os diagramas *Unified Modeling Language*, matriz de requisitos e documentação do projeto não são apresentados por confidencialidade e o escopo é somente a PrsP. Todos os programas envolvidos neste projeto usam a plataforma Microsoft .Net e a linguagem de programação utilizada é C# para o projeto Web e C++ para as aplicações desktop.

É importante que sejam bem definidas as relações de dependências técnicas em relação ao desenvolvimento de *software,* para viabilizar o trabalho simultâneo, com ganho de produtividade e sem esperas para iniciar o desenvolvimento de uma funcionalidade dependente de outra, o que por sua vez, pode ocasionar a perda de produtividade por negligência das dependências.

A equipe tem a seguinte formação multidisciplinar: programadores .net; especialistas em rede e banco de dados; programadores de linguagem C; testadores e designers UX. O trabalho é dividido em 2 equipes contendo 6 profissionais cada. As equipes são autogerenciáveis e multidisciplinares.

Cada equipe tem uma média histórica de velocidade de 144 *Story Points*, considerando um prazo de 2 semanas (10 dias úteis). As Story Points são contabilizadas através da sequência numérica de Fibonacci (1,3,5,8,11,13, 21..), essa estratégia possibilita descrever esforço como muito pequeno, pequeno, médio, grande e muito grande. O emparelhamento do projeto pela equipe foi avaliado e

pontuado de acordo com o histórico de *Story Points* da equipe em projetos anteriores, conforme mostra o quadro 5.

Quadro 5 – *Story Points* por *Release*

Release	***Story Points* (esforço)**
Cadastro	720
Visualizações essenciais	1008
Negociação	1008
Finanças	1008
Visualização de movimentações	1152
Análise de operadores	720
Visualizações para análise	864
Atendimento	288
Monitoramento	432
Total	**8.064**

Fonte: Elaborado pelo autor

O gráfico 1 considera o montante de 8.064 *Story Points* pontuado pela equipe, o emparelhamento de atividades é feito da seguinte forma: 8.064 / (2*144) = 56 semanas. Cada particionamento é representado por uma *release*, cada *release* tem duração máxima de 14 semanas. Ao total constam 9 entregas funcionais.

Gráfico 1 – Uso da PrsP versus programação em pares, por entrega (r*elease*)

Releases	**Semanas**																											
	2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**	**30**	**32**	**34**	**36**	**38**	**40**	**42**	**44**	**46**	**48**	**50**	**52**	**54**	**56**
Cadastro	X	X	X	X	X	O	O																					
Visualizações essenciais		O	O	O	O	X	X	O																				
Negociação					X	X	X	O	O	O	O																	
Finanças								X	X	X	X	O	O	O														
Visualizações Movimentações												X	X	X	O	O	O	O										
Análise de Operadores																		X	X	X	X	O	O					
Visualizações para análise																			O	O	O	X	X	X	X			
Atendimento																								O	O	X	X	
Monitoramento																									X	O	O	O

Fonte: Elaborado pelo autor

Legenda: “X” significa que a programação em pares é usada e, “O” a PrsP.

Conforme ilustrado no gráfico 1, metade do projeto foi desenvolvido com a técnica PrsP e a outra metade por meio da programação em pares.A metade do projeto feita com a programação em pares levou 1276,8 horas para que o desenvolvimento fosse concluído, enquanto a que foi feita com a PrsP levou 963,2 horas para desenvolver o mesmo número de *Story Points.*

As ferramentas de análise estática e dinâmica do código fonte, avisam sobre código fora do padrão, então o desenvolvedor refatora na hora. Ovalor do Desvio do Padrão é zero, pois neste projeto após o desenvolvimento, são realizadas inspeções de código e testes funcionais, *stress,* entre outros, devido à natureza crítica da aplicação, por envolver valores monetários de terceiros. Não houve registro defeitos após a implantação.

Os *Story Points* são uma métrica subjetiva, que não tem lastro com o código binário produzido pelo código fonte do *software*, é uma suposição feita pela equipe, portanto, não é 100% precisa*.* Não é coletada automaticamente, então a equipe gasta um tempo atribuindo os *Story Points* aos requisitos de *software*, porém, é uma maneira ágil e fácil para estimar e dividir o trabalho (meio a meio neste caso). Ajuda na fase 1 da PrsP a projetar o emparelhamento, inclusive das iterações simultâneas.

A métrica com lastro com o código binário captura com maior precisão a complexidade em relação à uma sem lastro com o código binário. Por ser coletada de forma automática é mais fácil, produtivo, confiável e preciso usá-la em comparação ao *Story Point*, porém, isso é feito depois que o código do programa está escrito, é melhor para medir a complexidade, mas não funciona para estimar, pois exige que o código do programa esteja pronto*.*

A Programação em pares obteve 37,81% da produtividade da PrsP por meio da métrica Produtividade direta.

A média de LOC por desenvolvedor por hora ao usar a PrsP é 169 e ao usar a programação em pares é 70. A densidade de defeitos é 0.

4.4 Terceiro exemplo de aplicação, *e-commerce*

Em 2010, a mesma equipe do exemplo 1, que participou do desenvolvimento do *software* de atividades esportivas, desenvolveu um sistema de *e-commerce*. Devido ao expressivo resultado obtido no projeto anterior, é adotada a Programação e revisão simultânea em Pares, como principal prática de desenvolvimento, o histórico de produtividade obtido por meio da contagem de linhas de código por

desenvolvedor, é usado como referência, porém, também são usadas neste exemplo de aplicação, as novas métricas usadas a partir do segundo exemplo de aplicação.

A fase 1 da PrsP, é o planejamento, a seleção dos pares, considerando a multidisciplinaridade e o projeto do emparelhamento das tarefas. A seleção dos pares para este exemplo, leva em consideração o estudo do Dyba et al. (2007), que a programação em pares não é vantajosa para programadores seniores, então os pares sempre são compostos por um desenvolvedor sênior e outro pleno. No projeto do emparelhamento das tarefas, são atribuídos *story points* aos requisitos de *software* e divididos 60% para o par usando a PrsP e 40% para o par usando a programação em pares, a disseminação de conhecimento é levada em consideração e são escolhidos os requisitos de forma colaborativa, contemplando as necessidades de aprendizado da equipe. A duração deste exemplo de aplicação é de 4 meses. São realizadas 4 *sprints* baseadas em valor, de duração variável, ao invés de prazo fixo, por exemplo de 2 semanas. A primeira *sprint* é de 3 semanas, a segunda é de 5 semanas, a terceira é de 3 semanas e a última é de 5 semanas.Na metade do projeto, após 8 semanas, são trocados os desenvolvedores plenos de equipe (rotação dos pares).

Conforme ilustra o gráfico 2, a cada *sprint* é trocada a prática de desenvolvimento do par, o par que estava usando a PrsP, passa a usar a programação em pares e vice-versa.

Gráfico 2 – Uso da PrsP versus programação em pares, por *Sprint* e por *Release*

Story Points	***Sprints***			
	1	**2**	**3**	**4**
350	**X** / **O**			
550		**O** / **X**		**O**
350			**X** / **O**	
550				**O** / **X**

Fonte: Elaborado pelo autor
Legenda: “X” significa que a programação em pares é usada e, “O” a PrsP.

Neste exemplo são escritos 89780 linhas de código e a densidade de defeitos é 5,3 por 10 mil LOC usando a PrsP. São escritos 39682 linhas de código e a densidade de defeitos é 7,2 por 10 mil LOC usando a programação em pares.

O descanso reflexivo/produtivo devido a lealdade, confiança, satisfação e comprometimento dos membros da equipe é feito de forma natural, por exemplo, durante um *happy hour* após o expediente um dos membros encontra um amigo pessoal (também programador) e aprende algo útil para este projeto e compartilha com a equipe, isso possivelmente contribui ao aumento da produtividade.

Dívida técnica é uma obrigação de manutenibilidade que a equipe acumula com suas ações (RAMASUBBU, 2015). Neste exemplo mesmo adotando o *mindset zero defect*, além do registro de defeitos, houve dívida técnica medida na forma de código fora do padrão, esta equipe não era exclusiva do projeto e a pressão do negócio era forte pela entrega rápida do *software*.

A métrica da produtividade direta, considera a complexidade do código desenvolvido e a degradação da qualidade é uma garantia que o código desenvolvido está aderente ao padrão de codificação e que a complexidade foi considerada ao medir a produtividade. A Programação em pares obteve 58,76% da produtividade da PrsP por meio da métrica Produtividade direta. A média obtida pela equipe neste exemplo de aplicação ao usar a PrsP é de 140 linhas de código por hora por desenvolvedor e ao usar a programação em pares é de 62 linhas de código por hora por desenvolvedor.

Uma queda de cerca de 4% na vantagem da PrsP em relação à programação em pares usando a métrica de contagem de linhas de código em comparação ao primeiro exemplo de aplicação.

4.5 Quarto exemplo, aplicativos financeiros

O primeiro aplicativo financeiro do mundo para Facebook® foi desenvolvido usando a PrsP. É uma versão reduzida de um *homebroker* para funcionar dentro da rede social, permitindo negociar ativos e visualizar as suas cotações. A equipe desde projeto é composta por 4 desenvolvedores seniores (2 especializados em programação e 2 em interface humano computador). Os *Slides* da apresentação feita num evento sobre engenharia de *software* acerca da nova técnica, contendo este exemplo de aplicação, está disponível na internet (KATTAN, 2013).

A duração deste exemplo de aplicação é de 6 meses. É feita apenas uma entrega contendo o aplicativo pronto. Um aplicativo financeiro por trabalhar com valores monetários de terceiros, requer um elevadíssimo nível de qualidade, um defeito pode sair caro. Neste exemplo, é fundamental o menor *time-to-market*

possível, por isso, é usada a PrsP (devido aos resultados positivos obtidos nos exemplos anteriores). Em 3912 horas é produzido 614184 linhas de código.

A média neste exemplo é de 157 linhas de código por hora por desenvolvedor.

O autor idealizou e desenvolveu em 2012 por meio da programação em pares um aplicativo de análise gráfica, que permite a um analista técnico do mercado de capitais, atribuir valores de suporte e resistência para cada ativo financeiro. Baseado nos valores atribuídos pelo analista técnico, em tempo real é capturada a cotação. Em seguida, um gráfico é atualizado com a recomendação de compra ou venda.

Durante 1280 horas são produzidas 120564 linhas de código. A média usando a programação em pares neste exemplo é de 74 linhas de código por hora por desenvolvedor.

4.6 Quinto exemplo, *software* financeiro para *Distributed Market Access*

Em 2013, a PrsP é usada de forma distribuída, os desenvolvedores não estavam no mesmo local, pois são aderentes ao *homeoffice*. A primeira fase da PrsP é feita no escritório de propriedade do autor, onde é feito o planejamento de maneira colaborativa, é acordado como dividir o desenvolvimento.

Na fase 4 da PrsP o desenvolvimento simultâneo é realizado em dependências físicas diferentes e quando querem se comunicarem usam o Skype®. Na fase 5 da PrsP conversaram via internet, antes de cada um revisar o trabalho do seu par, as ferramentas de análise de código, contribuíram de forma positiva na prevenção de erros.

O descanso reflexivo/produtivo da fase 7 da PrsP é estimulado pelo comprometimento dos desenvolvedores com o sucesso do projeto. A duração deste exemplo é de 9 meses. São feitas 2 iterações, a primeira após 5 meses do início do projeto e a segunda após 4 meses.

Na fase 1 da PrsP não foi detectado nenhum requisito que seria implementado usando a programação em pares ou o *mob programming*, porém, em algumas partes para disseminação de conhecimento técnico e operacional, o par decide implementar por meio da programação em par presencial no escritório com rotação da posse do teclado algumas vezes ao dia. Isso mostra que é uma equipe autogerenciável, responsável pelo trabalho e com autonomia para decisões.

Aproximadamente 5% deste projeto é implementado por meio da programação em pares, os outros 95% é com a PrsP. Então para fazer a comparação é considerada apenas a quantidade de horas usando PrsP equivalente a quantidade de horas usando a programação em pares. Durante 237 horas são produzidas 24174 linhas de código por meio da programação em pares e 37920 linhas de código por meio da PrsP. A média neste exemplo é de 102 LOC/hora por desenvolvedor ao usar a programação em pares e 160 LOC/hora por desenvolvedor com a PrsP.

4.7 Sexto exemplo, *software* para seguradoras

Em 2014, na cidade de Lisboa em Portugal, é usada a PrsP num projeto de Seguros, um desenvolvedor teve uma necessidade médica e ausentar-se-ia por uma semana, havia a necessidade da disseminação do conhecimento, porém a produtividade para este projeto é um fator crucial, os desenvolvedores já conheciam o trabalho do autor e acharam que seria útil usar a PrsP para disseminar o conhecimento sem perder a produtividade. Os desenvolvedores relataram aumento perceptível da produtividade e ao avaliar-se a produtividade obtida em comparação ao histórico da equipe constata-se aumento da produtividade ao usar a PrsP, sem perder a qualidade do produto de *software,* além do benefício da transferência de conhecimento para o outro desenvolvedor realizar sua intervenção médica tranquilamente.

A densidade de defeito neste exemplo é 0. A equipe tem um histórico de defeitos usando a programação em pares de 1 defeito por 12 mil linhas de código. Durante 87 horas são produzidas 5742 linhas de código por meio da programação em pares e 12354 linhas de código por meio da PrsP. A média de LOC/desenvolvedor por hora é 66 com a programação em pares e 142 com a PrsP.

4.8 Sétimo exemplo, gerenciador de contratos

Em 2015 está sendo desenvolvido um gerenciador de contratos usando a PrsP. Na fase 1 da PrsP não foi detectado nenhum requisito que seria implementado usando a programação em pares ou o *mob programming*, porém, em algumas partes para disseminação de conhecimento técnico e operacional, o par decide implementar por meio da programação em par. Aproximadamente 5% deste projeto é implementado por meio da programação em pares, os outros 95% é com a PrsP.

Então para fazer a comparação é considerada apenas a quantidade de horas usando PrsP equivalente a quantidade de horas usando a programação em pares.

Durante 237 horas são produzidas 24174 linhas de código por meio da programação em pares e 37920 linhas de código por meio da PrsP. A média neste exemplo é de 102 LOC/hora por desenvolvedor ao usar a programação em pares e 160 LOC/hora por desenvolvedor com a PrsP.

4.9 Oitavo exemplo, exercício de programação

Entre 2014 e 2015, a PrsP é usada em ambiente acadêmico, em cursos de programação, os programadores são classificados como juniores, com base nos resultados da aplicação de um questionário *online* (HEREZ, 2015), a complexidade é baixa dos requisitos de *software* referentes ao exercício de programação. A fase 7 descanso produtivo e resolução de conflitos é omitida, pois o exercício tem duração de apenas uma aula.

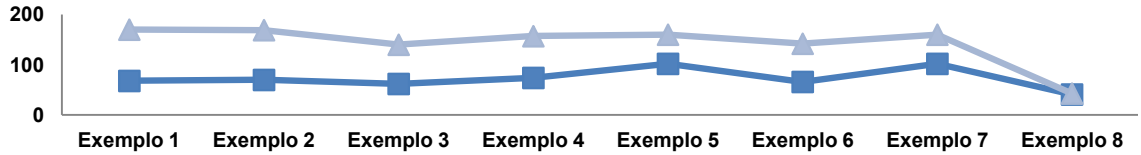
É proposta a participação neste exemplo aos alunos no último dia de aula da matéria de programação do último ano do cursos de ciências da computação. É feito uma aula no decorrer do curso de programação sobre a PrsP e programação em pares. É proposto o desenvolvimento de um aplicativo genérico contendo cadastros, relatórios, controle de acesso, *logs*, regras de negócios hipotéticas para haver dependências entre as funções. São formados 4 grupos de 4 alunos. Cada grupo divide de maneira colaborativa em 2 partes a serem desenvolvidas uma metade usando a PrsP e a outra usando a programação em pares. Em 5 horas são produzidas 205 linhas de código por meio da programação em pares e 215 linhas de código por meio da PrsP.

A média neste exemplo é de 41 LOC/hora por desenvolvedor ao usar a programação em pares e 43 LOC/hora por desenvolvedor com a PrsP. A densidade de defeitos é medida por KLOC, usando a programação em pares é obtida uma média de 2,5 defeitos a cada cem linhas de código e usando a PrsP 2 defeitos a cada cem linhas de código.

4.10 Resultados

Conforme ilustrado no gráfico 3, a média de linhas de código produzidas por desenvolvedor por hora, indica que a PrsP é mais produtiva que a programação em pares nos 8 exemplos.

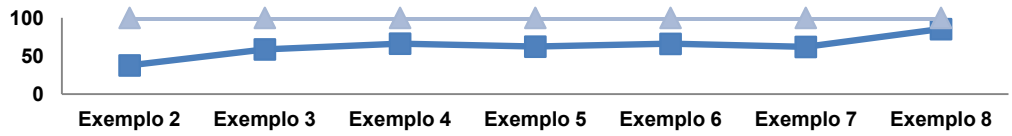
Gráfico 3 –LOC/desenvolvedor/hora média da programação em pares versus PrsP



Fonte: Elaborado pelo autor
Legenda: Quadrado é média da programação em pares e triângulo da PrsP

Conforme ilustrado no gráfico 4, a métrica produtividade direta, nos 7 exemplos que é usada aponta para a mesma direção que LOC/desenvolvedor por hora, ambas indicam que a PrsP é mais produtiva que a programação em pares.

Gráfico 4 –Produtividade direta da programação em pares versus PrsP



Fonte: Elaborado pelo autor
Legenda: Quadrado é média da programação em pares e triângulo da PrsP

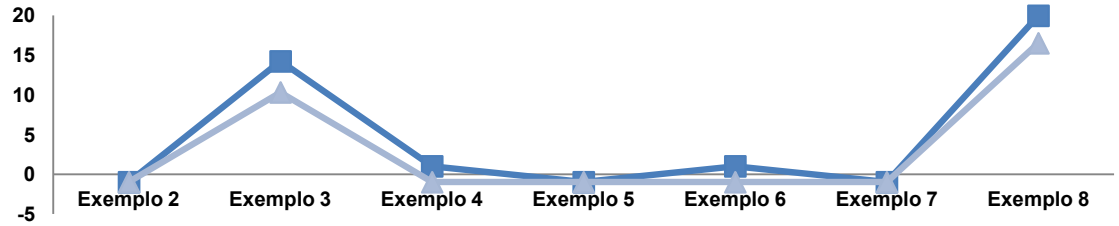
Conforme ilustrado no gráfico 5, a densidade de defeito a cada 10 mil linhas de código nos 8 exemplos de aplicação aponta que no mínimo a PrsP produziu código com a qualidade equivalente à programação em pares.

Gráfico 5 –Densidade de defeitos por 10KLOC da programação em pares e PrsP

Fonte: Elaborado pelo autor
Legenda: Quadrado é média da programação em pares e triângulo da PrsP

Conforme ilustrado no gráfico 6, a métrica degradação da qualidade, nos 7 exemplos em que foi empregada, indica que a PrsP produziu código de qualidade superior em relação à programação em pares.

Gráfico 6 –Degradação da qualidade da programação em pares versus PrsP



Fonte: Elaborado pelo autor
Legenda: Quadrado é média da programação em pares e triângulo da PrsP

5 ANÁLISE DOS RESULTADOS E VALIDAÇÃO DA NOVA TÉCNICA

Para validação da nova técnica são analisados os resultados dos exemplos de aplicação descritos. É realizada também uma análise comparativa dos resultados das métricas. Os dados dos exemplos de aplicação são coletados ao longo dos últimos oito anos. A análise destes dados busca por explicações, inicia-se combinando pontos chaves nos dados coletados. São escritos textos curtos referentes a estes pontos chaves e são classificados nas categorias: positivo e/ou negativo. O Quadro 6 contém os pontos chaves encontrados, escritos na forma de um texto curto, porém, ainda não estão classificados como positivo e/ou negativo, bem como, uma breve descrição sobre eles.

Quadro 6 – Pontos chaves

Pontos chave	Descrição
Primeira fase destinada ao planejamento, a realização da seleção dos pares, ao projeto do emparelhamento e iterações simultâneas.	Tipicamente ao usar a programação em pares no máximo considera-se a seleção dos pares. A PrsP vai além disso, quanto maior for o conhecimento do domínio do problema, melhor projeta-se o emparelhamento e se houver mais de um par, pode-se considerar realizar as iterações (*sprints)* de maneira simultânea.
Não senta-se lado a lado durante todo o expediente	Diferentemente da programação em pares, na PrsP não se senta lado a lado durante todo o expediente.
Sempre na posse do computador	Diferentemente da programação em pares, na PrsP o desenvolvedor sempre está na posse do teclado, mouse e computador. Observou-se que estimula o comprometimento.
Conversa-se quando necessário e útil	Uma grande mudança em comparação a programação em pares é o fato de não sentar lado a lado e conversa-se somente quando necessário.
Desenvolvimento paralelo/simultâneo	O trabalho é executado em paralelo e quanto maior for o conhecimento do domínio do problema, mais é possível projetar o emparelhamento das atividades de maneira simultânea, quando há mais de um par na equipe.
Disseminação do conhecimento técnico e operacional	Na fase 1 da PrsP é decidido pela equipe se algum requisito será feito usando o *mob programming,* ou se será feito um *coding dojo.* No exemplo 5 usou-se a programação em pares.
Útil para código críticos e complexos	Sim, nos exemplos de aplicação financeira ao analisar-se os resultados, observa-se que a PrsP funcionou melhor que a programação em pares.
Útil para códigos simples	Sim, o oitavo exemplo é de baixa complexidade e a PrsP apresentou maior produtividade e qualidade em comparação a programação em pares.

Continua na próxima página

Pontos chave	Descrição
Útil para códigos de complexidade média	Sim, os exemplos 1, 3 e 7 são de complexidade média e a PrsP apresentou maior produtividade e qualidade em relação à programação em pares.
Revisão de código em pares	É feita na fase 5 da PrsP, diferentemente da programação em pares, que faz no mesmo instante, pois o par por está do lado.

Fonte: Elaborado pelo autor

O Quadro 7 contém os pontos chaves classificados e submetidos a uma comparação constante, a classificação foi analisada novamente a cada inclusão de um novo ponto chave, para verificar se era conflitante, alterava ou tinha relação com algum outro. Tem ponto chave que é classificado como positivo e negativo, então é colocada uma justificativa do motivo pelo qual é classificado nas duas categorias.

Quadro 7 – Pontos chaves classificados nas categorias: positivo e/ou negativo

Pontos chave	Categorização	
Não senta-se lado a lado durante todo o expediente	Positivo para a execução em paralelo/simultâneo. Positivo para o aumento da produtividade. Positivo para o aumento da satisfação, motivação, colaboração, comunicação e confiança entre os membros da equipe. (somente no primeiro e oitavo exemplo de aplicação foi possível coletar dados sobre a moral da equipe usando um calendário *niko-niko (*NIKO-NIKO, 2015)	Negativo para a comunicação presencial instantânea e avisar sobre um detalhe ou maneira melhor para um trecho de código imediatamente sem ter que esperar pela revisão na Fase 5 da PrsP. Negativo para um programador experiente treinar um novato.
Sempre na posse do computador	Positivo para a execução em paralelo/simultâneo. Positivo para aumentar o comprometimento. Positivo para o aumento da satisfação, motivação, colaboração, comunicação e confiança entre os membros da equipe. (somente no primeiro e oitavo exemplo de aplicação foi possível coletar dados sobre a moral da equipe usando um calendário *niko-niko (*NIKO-NIKO, 2015)	
Conversa-se quando necessário e útil	Positivo, para produtividade e moral da equipe.	
Desenvolvimento paralelo/simultâneo	Positivo, para produtividade.	
Disseminação do conhecimento técnico e operacional	Positivo, trabalho em pares estimula a comunicação e disseminação do conhecimento e a fase 1 da PrsP permite que se planeje o uso de *mob programming, coding dojo,* programação em pares ou qualquer técnica correlata.	

Continua na próxima página

Pontos chave	Categorização
Primeira fase destinada ao planejamento, a realização da seleção dos pares, ao projeto do emparelhamento e iterações simultâneas.	Positivo, quanto maior for o conhecimento do domínio do problema, melhor projeta-se o emparelhamento e se houver mais de um par, pode-se considerar realizar as iterações (*sprints)* de maneira simultânea com o principal objetivo de aumentar a produtividade. A seleção dos pares contribui para a disseminação do conhecimento técnico e operacional.
Diversos níveis de complexidade	Positivo, a PrsP é útil para algoritmos de diversos níveis de complexidade.

Fonte: Elaborado pelo autor

Uma possível explicação para a queda na produtividade de cerca de 4% no terceiro exemplo em relação ao primeiro exemplo de aplicação, é que o primeiro é uma migração e o terceiro é um novo desenvolvimento, portanto, tem mais incerteza em relação ao primeiro, onde os requisitos eram bem conhecidos pela equipe.

A PrsP é vantajosa em relação à programação em pares, independente da experiência do programador e complexidade da tarefa, nos 8 exemplos realizados. Porém, em nenhum dos 8 exemplos teve um caso de programador experiente treinando um novato. A programação em pares provavelmente é melhor que a PrsP neste caso, ou ter seu uso determinado para este caso, na fase 1 da PrsP, como é feito para o *mob programming* dependendo o requisito, ou escolher fazer um *coding dojo*, então uma possibilidade é escolher implementar um requisito usando programação em pares para um desenvolvedor experiente treinar um novato.

A produtividade do desenvolvimento foi superior para a produção de algoritmos de todos os níveis de complexidade em relação à programação em pares nos 8 exemplos de aplicação. A qualidade do produto foi no mínimo equivalente à programação em pares, com todas as métricas usadas nos 8 exemplos.

No primeiro e oitavo exemplo foi possível coletar por meio de um calendário *niko-niko (*NIKO-NIKO, 2015) dados sobre a satisfação, motivação, colaboração, comunicação e confiança entre os membros da equipe. Uma possível explicação para a preferência pela PrsP em detrimento a programação em pares é o fato de não precisar sentar lado a lado durante todo expediente e sempre estar na posse do teclado. Ao observar este fenômeno durante a condução dos exemplos o autor lembrou que quando começou a programar em 1985 os computadores eram caros e os cursos não permitiam ter 1 aluno por computador, e a posse do teclado era uma baita felicidade, doou todos seus brinquedos, exceto um TK95, um micro-computador com as linguagens de programação *assembler e basic,* que adorava

poder programar nele e mostrar à sua mãe os aplicativos, desenhos, músicas e jogos programados. A posse do teclado para um programador tem um efeito psicológico que merece ser melhor estudado. A PrsP é como jogar *online*, se tem a posse do teclado com a vantagem de não estar sozinho.

Dívida técnica é uma obrigação de manutenibilidade que a equipe acumula com suas ações (RAMASUBBU, 2015). No terceiro exemplo, nota-se mesmo adotando o *mindset zero defect* na PrsP, além do registro de defeitos, houve dívida técnica medida na forma de código fora do padrão, esta equipe não era exclusiva do projeto e a pressão do negócio era forte pela entrega rápida do *software*. A métrica degradação da qualidade e o *mindset zero defect* incorporado à PrsP ajudaram a evidenciar a dívida técnica e contribuíram para a sua eliminação nos exemplos 1,2,4,5 e 6, devido às ferramentas de análise de código usadas avisarem sobre os códigos fora do padrão de maneira imediata. Dependendo da pressão do negócio e do tipo de aplicativo, pode-se correr o risco e refatorar o código posteriormente, porém, recomenda-se que faça-se a refatoração no mesmo instante.

A métrica degradação da qualidade ajudou na avaliação da aderência dos algoritmos produzidos ao padrão de codificação. A métrica produtividade direta ajuda a considerar a complexidade do código e aderência dos algoritmos ao padrão de codificação na medição da produtividade, indo além da contagem do número de linhas de código e da densidade de defeitos. *Story Points* é uma métrica subjetiva, não tem lastro com o código binário produzido pelo código fonte do *software*, é uma suposição feita pela equipe, portanto, não é 100% precisa. Não é coletada automaticamente, então a equipe gasta um tempo atribuindo os *Story Points* aos requisitos de *software*, porém, é uma maneira ágil e fácil para estimar e dividir o trabalho o que ajuda na fase 1 da PrsP no projeto do emparelhamento, inclusive para *sprints* simultâneas. A métrica complexidade tem com lastro com o código binário, por isso, captura com maior precisão a complexidade em relação à uma sem lastro com o código binário. Por ser coletada de forma automática é mais fácil, produtivo, confiável e preciso usá-la em comparação ao *Story Point*, porém, isso é feito depois que o código do programa está escrito, é melhor para medir a complexidade, mas não funciona para estimar, pois exige que o código do programa esteja pronto. Quanto maior domínio do problema a ser resolvido se tiver, maior é a vantagem de usar *Story Points* na primeira fase da PrsP para projetar o emparelhamento e iterações (*sprints*) simultâneas, no segundo exemplo usar os

Story Points para esta finalidade funcionou muito bem. O quadro 8 mostra as análises baseadas nos resultados obtidos de todas as novas métricas propostas e acerca do uso de *Story Points*, como uma possibilidade para servir de apoio à fase 1 da PrsP para projetar o emparelhamento das iterações (*sprints)* simultâneas, por meio do particionamento de suas funções, estimadas por *Story Points* e divididas pela equipe com base nesta estimativa para balancear a divisão das partes a serem desenvolvidas.

Quadro 8 – Análise sobre as métricas baseadas nos resultados obtidos

Métrica	**Análise**
Desvio do Padrão = Número total de linhas fora do padrão / TLOC	As ferramentas de análise de código atuais permitem configurar um padrão de codificação e exibir um relatório acerca das linhas de códigos fora do padrão.
Degradação da Qualidade = Desvio do Padrão + Densidade de Defeito X 2	Além da densidade dos defeitos, considera também o fator de teste e o desvio do padrão de codificação.
Complexidade = Complexidade Ciclomática – LCOM + DIT + NOC + AC +EC	Além da complexidade ciclomática, considera o acoplamento aferente e eferente, falta de coesão dos métodos, número de filhos e profundidade na árvore de herança.
Produtividade direta = (TLOC X Complexidade) X Degradação da Qualidade	Além das linhas de código, é considerada a complexidade deste código produzido e sua qualidade (precisão superior a contagem de defeitos, pois considera também: padrão de codificação e fator de testes). Pelo fato de considerar a complexidade e qualidade, sua precisão para medir a produtividade é superior a simples contagem das LOC.
Story Points	Na fase 1 da PrsP foi eficaz para estimar o esforço e projetar o emparelhamento das iterações simultâneas, atribuir os pontos por meio de uma sequência Fibonacci (1,3,5,8,11,13, 21..), observou-se uma maneira eficaz para contabilizar o esforço em muito pequeno, pequeno, médio, grande e muito grande. Uma possibilidade detectada durante a execução dos exemplos e que pode ser útil na fase 1 da PrsP é a realização de uma prática ágil chamada *planning poker,* onde são atribuídos os *Story Points* com auxílio de um baralho, ideal que este baralho tenha valores de uma sequência Fibonacci.

Fonte: Elaborado pelo autor

6 CONCLUSÃO, LIMITAÇÕES E TRABALHOS FUTUROS

A PrsP foi eficaz nos oito exemplos de aplicação, onde é observado aumento da produtividade no desenvolvimento e qualidade do produto de *software*, de acordo com todas as métricas usadas em relação à programação em pares.

As métricas degradação da qualidade e produtividade direta são consideradas efetivas nos 8 exemplos de aplicação na avaliação da aderência do algoritmo produzido ao padrão de codificação. No exemplo 3, houve dívida técnica, devido à pressão por prazo, mesmo com à priorização da qualidade na PrsP, não foi possível eliminá-la, porém, foi evidenciada pelas novas métricas.

A PrsP contribuiu para reduzir e evidenciar a dívida técnica no terceiro exemplo de aplicação e eliminá-la em todos os outros 7 exemplos.

Observa-se que há uma relação entre a PrsP e a dívida técnica, esta relação é uma das sugestões para trabalhos futuros, devido à limitação na validade da conclusão que a PrsP reduz e evidência a dívida técnica, podendo eliminá-la, pois esta pesquisa realizou apenas 8 exemplos de aplicação, em 4 empresas de capital privado, envolvendo somente 55 desenvolvedores e dentre eles o próprio autor.

A PrsP foi efetiva também para programadores experientes, ao contrário da programação em pares, que funciona melhor para programadores juniores.

O quadro 9 é uma diretriz baseada nos 8 exemplos de aplicação realizados. Além da experiência do programador, leva-se em consideração a combinação da experiência do par. São considerados para determinar se é vantajoso usar a PrsP os seguintes objetivos: produtividade, qualidade e moral da equipe (satisfação, motivação, colaboração, comunicação e confiança entre os membros da equipe).

Quadro 9 – Diretriz para quando usar a PrsP

Experiência do par	**Complexidade da tarefa**	**É vantajoso usar a PrsP?**
Júnior e Júnior	Baixa	Sim, o aumento da produtividade é o principal ganho, observa-se a melhora da qualidade e aumento da moral da equipe no exemplo 8, onde a complexidade da tarefa é baixa. Uma limitação da pesquisa é que teve apenas um exemplo de aplicação de complexidade baixa que usou pares com a combinação de experiência: júnior e júnior.
	Média	
	Alta	
Júnior e Pleno	Baixa	Nenhum dos 8 exemplos realizou tal combinação, são realizadas apenas as combinações: sênior com sênior, sênior com pleno e júnior com júnior.
	Média	
	Alta	

Continua na próxima página

Experiência do par	Complexidade da tarefa	É vantajoso usar a PrsP?
Pleno e Pleno	Baixa	Nenhum dos 8 exemplos realizou tal combinação, são realizadas apenas as combinações: sênior com sênior, sênior com pleno e júnior com júnior.
	Média	
	Alta	
Pleno e Sênior	Baixa	Sim, o aumento da produtividade é o principal ganho, além da melhora na qualidade do produto e a disseminação do conhecimento.
	Média	
	Alta	
Sênior e Júnior	Baixa	Nenhum dos 8 exemplos realizou tal combinação, são realizadas apenas as combinações: sênior com sênior, sênior com pleno e júnior com júnior. Baseando-se nos resultados deste trabalho, pressupõe-se que a combinação sênior com júnior, em relação à produção de linhas código aderentes ao padrão de codificação e menor densidade de defeitos, seria melhor a PrsP em comparação à programação em pares, porém, em relação à aprendizagem a programação em pares seria melhor em relação à PrsP. Sugere-se a averiguação desta hipótese como sugestão para trabalhos futuros.
	Média	
	Alta	
Sênior e Sênior	Baixa	Sim, o aumento da produtividade é o principal ganho, além da melhora na qualidade do produto e a disseminação do conhecimento.
	Média	
	Alta	

Fonte: Elaborado pelo autor

A principal limitação desta pesquisa é o número reduzido de exemplos de aplicação, que obriga a conclusão basear-se em apenas 8 exemplos que envolvem apenas 55 desenvolvedores e nestes exemplos não foi possível realizar todas as combinações possíveis em relação ao nível de experiência do desenvolvedor dentro do par e complexidade da tarefa. Uma possibilidade a ser averiguada na fase 1 da PrsP com maior profundidade, é usar a programação em pares da mesma maneira como no exemplo 1 ao usar o *mob programming* para disseminação de conhecimento, para se o principal objetivo for o aprendizado de um par composto por um sênior e júnior, se é possível incorporá-la na fase 1 da PrsP, na seleção dos pares e projeto do emparelhamento.

Como sugestões para trabalhos futuros: realizar experimentos e estudos de caso, visando averiguar se a nova técnica preserva o aumento da satisfação, motivação, aprendizado, companheirismo, confiança e capacidade técnica da equipe em comparação às técnicas correlatas. Se evita a dívida técnica e a evasão dos talentos da equipe e da empresa. Se o custo econômico financeiro ao usar a PrsP é vantajoso em comparação à outras técnicas. Realizar experimentos, estudos de caso e exemplos de aplicação das métricas propostas por este trabalho para produtividade no desenvolvimento de *software* e qualidade do produto de *software.*

REFERÊNCIAS

ADAMS, S. **Figura 3.** DILBERT © 2003 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved. Disponível em: http://dilbert.com/strip/2003-01-09 Acessado em: 04/03/2014

ADAMS, S. **Figura 4.** DILBERT © 2003 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved. Disponível em: http://dilbert.com/strip/2003-01-11 Acessado em: 04/03/2014

ALI, M.; BROOKS, L. **A situated cultural approach for cross-cultural studies**, Journal of Enterprise Information Management. Bingley, v.22, n.5, p. 548-563, 2009

ANDRADE, S. **Abordagem dirigida ao gerenciamento do conhecimento para a melhoria dos fatores de produtividade do desenvolvimento de software ágil XP.** 112p. Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software. São Paulo, 2012.

ANTINYAN, V.; STARON, M.; DEREHAG, J.; RUNSTEN, M.; WIKSTRÖM, E.; MEDING, W.; HENRIKSSON, A.; HANSSON, J. **Identifying Complex Functions By Investigating Various Aspects of Code Complexity.** Science and Information Conference July 28-30, 2015 | London, UK.

ARISHOLM, E.; GALLIS, H.; DYBA, T.; SJOBERG, D. **Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise**, IEEE Transactions on Software Engineering, vol. 33, no. 2, 2007, pp. 65–86.

BANDUKDA, M.; NASIR, Z. **Efficacy of distributed pair programming.** IEEE, 2010

BECK, K. Publicação feita em seu perfil **@KentBeck** no Twitter. Compartilhado no perfil @HerezKattan, 2012.

BECK, K. **Embracing change with extreme programming.** *IEEE Computer*, páginas 70–77, Outubro de 1999.

BECK, K. **Extreme Programming Explained: Embrace Change**. 1. ed. Boston, Estados Unidos. Addison-Wesley, 2000. 58p.

BECK, K.; BEEDLE, M.; BENNEKUM, A.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT,A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto for agile software development.** Disponível em: **www.agilemanifesto.org**, 11/02/2001. Acessado em: 11/02/2015.

BECK, K. **Test Driven Development: By Example.** Addison-Wesley Professional, primeira edição, Boston-USA, 2003.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. segunda edição, Boston-USA. Addison-Wesley, 2004. 75p.

BECK, K.; FOWLER, M. **Planning Extreme Programming**. 1. ed. Boston: Addison-Wesley Professional, 2001.

BEGEL, A.; NAGAPPAN, N. **Pair programming: what's in it for me?** In *ESEM '08: Proc. of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 120–128, New York, NY, USA, 2008. ACM.

BELLA, E.; FRONZA, I.; PHAPHOOM, N.; SILLITTI, A.; SUCCI, G.; VLASENKO, J. **Pair Programming and *Software* Defects — A Large, Industrial Case Study,** IEEE Transactions on Software Engineering, VOL. 39, NO. 7, JULY 2013.

BENNETT, J. G.; LAMB, T. Concurrent engineering : application and implementation for U.S. shipbuilding. In: SHIP PRODUCTION SYMPOSIUM, Seattle, 1995. **Proceedings**. Jersey City, NJ. : SNAME, 1995. p.23.1-23.4.

BOEHM, B. et al. **Productivity Trends in Incremental and Iterative Software Development**. In Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09), 2009, Lake Buena Vista, Florida, USA, p. 1-10.

BORGES, M. **Suporte por Computador ao Trabalho Cooperativo.** Jornada de Atualização em Informática: Congresso Nacional da SBC. Canela, Brasil, 1995.

BROOKS, F. ***Mythical Man Month: Essays on Software Engineering*.** Reading, Massachusetts-USA, Addison-Wesley Professional, 1975.

CARTER, D. E.; BAKER, B. S. **Concurrent engineering: the product development environment for the 1990s.** Addison Wesley, 1991.

CARVER, J.; CAGLAYAN, B.; HABAYEB,M.; PENZENSTADLER,B.; YAMASHITA,A. **Collaborations and Code Reviews.** IEEE Software, September/October 2015.

CHOI, K. **Evaluating Gender Significance Within a Pair Programming Context.** 46th Hawaii International Conference on System Sciences, IEEE Computer Society, 2013

CHONG, J.; HURLBUTT, T. **The Social Dynamics of Pair Programming**, in *International Conference on Software Engineering (ICSE) 2007*, Minneapolis, MN, 2007, pp. 354-363

COCKBURN, A.; WILLIAMS, L. **The Costs and Benefits of Pair Programming**, in eXtreme Programming and Flexible Processes in Software Engineering, XP2000, Cagliari, Italy, June 2000.

COCKBURN, A.; WILLIAMS, L. **The costs and benefits of pair programming, *in extreme programming examined.,*** Boston, Addison-Wesley, 2001.

CODEFLOW, Disponível em: <http://visualstudioextensions.vlasovstudio.com/2012/01/06/codeflow-code-review-tool-for-visual-studio/>, Acessado em:18/11/2015

COLLABORATOR, Disponível em: <http://smartbear.com/products/software-development/code-review/>, Acessado em:18/11/2015

CONSTANTINE, L. ***Constantine on Peopleware***, Yourdon Press, Englewood Cliffs, N.J., 1995.

COPLIEN, J.; SCHMIDT, D. **A Development Process Generative Pattern Language.** *Pattern Languages of Program Design*, Addison-Wesley, Reading, Mass., 1995, pp. 183–237.

COPLIEN, J.; HARRISON, N. **Organizational Patterns of Agile Software Development.** Upper Saddle River, NJ: Pearson Prentice-Hall, 2005.

CRITICS, Disponível em: <https://sites.google.com/a/utexas.edu/critics/>, acessado em: 11/10/2015.

CRUZ, T. **Workflow: A Tecnologia que vai revolucionar os Processos**. Editora Atlas, 2000.

CZERWONKA, J.; GREILER, M.; TILFORD, J. **Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down.** 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. ICSE 2015, Florence, Italy. © IEEE 2015

DEBONI, J.E.Z.; MARTINI, J.S.C. Gerenciamento de documentos em ambiente de engenharia concorrente. In:CONGRESSO INTERNACIONAL DE COMPUTACAO GRAFICA, 5-7 de abril, 1994, São Paulo, SP. **IPEN-DOC 05773**

DU, W.; OZEKI, M.; NOMIYA, H.; MURATA, K.; ARAKI, M. **Pair programming for enhancing communication in the fundamental C language exercise.** *Kyoto Insititute of Technology Kyoto, Japan* 2015 IEEE 39th Annual International Computers, Software & Applications Conference © 2015 IEEE

DYBA, T.; ARISHOLM, E.; SJOBERG,K.; HANNAY,E.; SHULL,F. **Are Two Heads Better than One? On the Effectiveness of Pair Programming.** IEEE *Software*, vol. 24, no. 6, 2007, pp. 12–15.

DYBA, T.; DINGSØYR, T. **Empirical studies of agile software development: A systematic review.** Elsevier, available on www.sciencedirect.com. Information and Software Technology 50 (2008). Pages 833-859. Trondheim, Norway, 02/02/2008

ECLIPSE FOUNDATION . **Eclipse**: integrated development environment. Version 4.2., 2012. Disponível em: <https://www.eclipse.org/>. Acesso em: 14/06/2015.

ECLIPSE METRICS, *Plug-in do Eclipse Metrics.* Disponível em: <http://metrics.sourceforge.net/>. Acesso em: 14/06/2015.

ESTÁCIO, B.; PRIKLADNICKI, R. **Distributed Pair Programming: A Systematic Literature Review.** Information and Software Technology 63 (2015) 1–10. Contents lists available at ScienceDirect © 2015 Elsevier B.V. All rights reserved.

FRASER, S.; et al. **Culture and Agile: Challenges and Synergies.** In: Agile Processes in Software Engineering and Extreme Programming. September 2008, Limerick. Proceedings… Berlin: Springer, 2008 pp. 251-255.

FREUDENBERG, S.; ROMERO, P.; BOULAY, B. **'Talking the talk': Is Intermediate-level conversation the key to the pair programming success story?**, in *Agile 2007*, Washington, DC, 2007, pp. 84-91.

GALDÁMEZ, E. **Integrando os Recursos Humanos com Engenharia Simultânea.** Universidade de São Paulo, São Paulo, 2000.

GATES, B. **Lines of Code** Editado em: June 29, 2014. Acessado em: 09/11/2015. Disponível em: <http://c2.com/cgi/wiki?LinesOfCode>.

GERRIT, Disponível em: <http://code.google.com/p/gerrit>, Acessado em:18/11/2015

GIT*,* Disponível em: <https://git.kernel.org>, Acessado em: 14/07/2015

GOUVEIA, L. M. Borges (2002). CSCW – Trabalho Cooperativo Suportado por Computador, acessado em 17/02/15 e disponível na internet em: http://www2.ufp.pt/~lmbg/formacao/group_cscw.PDF

HAMMER, M. **Re-engineering Work: Don't Automate, Obliterate,** *Harvard Business Review*, July-August, pp. 104-112. Cambridge, Massachusetts, 1990

HAMMER, M.; CHAMPY, J. **Reengenharia revolucionando a empresa em função dos clientes, da concorrência e das grandes mudanças da gerência.** Editora Campus, Rio de janeiro, RJ, 1994. Tradução da obra original: **Reengineering the corporation, a manifesto for business revolution.** New York, NY – USA, 1993.

HAUPTMAN, O., HIRJI, K., **The influence of process on project outcomes in product development: an empirical study of cross-functional teams***,* IEEE Transactions on Engineering Management, volume 43, n. 2, p. 153-163, May, 1996.

HARTLEY, J. **Concurrent Engineering: shortening lead times, raising quality and lowering costs**, Cambridge, Massachusetts, Productivity Press, 1992.

HAWRYSZKIEWYCZ, I. Designing the Networked Enterprise, Artech House, 1997.

HEREZ, questionário de aderência à programação em pares, disponível em: <http://herez.net/Questionario>, acessado em: 10/10/2015

HÖFER, A. **Video Analysis of Pair Programming**, in *Workshop on Scrutinizing Agile Practices at the International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 37-41.

HOHMAN, M.; SLOCUM, A. **Mob Programming and the Transition to XP.** Disponível em: <http://cf.agilealliance.org/articles/system/article/file/998/file.pdf> Chigado – IL / USA, agosto/2001. Acessado em: 07/09/2015.

HULKKO, H; ABRAHAMSSON, P. **A Multiple Case Study on the Impact of Pair Programming on Product Quality**, in *International Conference on Software Engineering (ICSE) 2005*, St. Louis, Missouri, USA, 2005, pp. 495-504.

JANIS, I. **Groupthink: Psychological Studies of Policy Decisions and Fiascoes** 2nd Edition. 349 pages. Publisher: Cengage Learning; 2 edition (May 19, 1982) ISBN-10: 0395317045

JONES, D.; FLEMING, S. **What Use Is a Backseat Driver? A Qualitative Investigation of Pair Programming** 2013 IEEE Symposium on Visual Languages and Human-Centric Computing © IEEE 2013.

KAIZENWORLD; **One-Piece Flow**. Disponível em: http://www.kaizenworld.com/kaizen/one-piece-ow.html acessado em:07/09/2015.

KATTAN, H. M. Programação simultânea em pares. Uma nova técnica ágil: Programação em pares evoluída pela Engenharia Simultânea. In: AGILE TRENDS, 2013, São Paulo, SP. **Eletronic Proceedings**... Disponível em: <http://herez.net/palestras> ou <http://agiletrendsbr.com/2013/programacao/>; Acesso em: 11 fevereiro de 2015.

KATZENBACH, J.; SMITH D. **The Wisdom of Teams**, Harvard Business School Press. 1993

KATZENBACH, J.R., SMITH, D.K. **The discipline of teams**, Harvard Business Review, Boston, March-April 1993, pp. 111-124.

KON, F.; MELO, C.; CRUZES, D.; CONRADI, R. **Interpretative Case Studies on Agile Team Productivity and Management.** Information and Software Technology, volume 55, issue 2, fevereiro de 2013, pp. 412-427.

KREBS, W. Turning the knobs: A coaching pattern for XP through agile metrics. XP/Agile Universe 2002, Lecture Notes on Computer Science 2418:60–69, 2002. 67, 68,81, 82, 83, 84, 85, 101, 113

KRUGLIANSKAS, I. **Engenharia Simultânea: organização e implantação em empresas brasileiras.** Trabalho apresentado no XVII Simpósio Nacional de Gestão da Inovação Tecnológica, publicado na revista de administração, São Paulo – SP volume 28, número 4, páginas 104-110. Dezembro/1993

MANDHAN**,** N.; VERMA, D.; KUMAR, S. **Analysis of Approach for Predicting Software Defect Density using Static Metrics.** International Conference on Computing, Communication and Automation (ICCCA2015) © IEEE, 2015.

MARTINI, J.S.C.; DEBONI, J.E.Z. AEC and concurrent engineering. In: PDMS USERS GROUP ASSOCIATION MEETING, June, 1995,. **IPEN-DOC 05805**

MARTINI, J.S.C.; DEBONI, J.E.Z. Aplicação da engenharia simultânea em projetos de AEC - realidade e perspectivas. In: SIMPOSIO INTERNACIONAL DE COMPUTACAO GRAFICA,11 de abril, 1995, São Paulo, SP. **IPEN-DOC 05806**

MCAVOY, J.; BUTLER, T. **The impact of the Abilene Paradox on double-loop learning in an agile team.** Information and Software Technology Páginas 552-563 Volume 49 Issue 6, Butterworth-Heinemann Newton, MA, USA, June, 2007

MCDOWELL, C.; WERNER, L.; BULLOCK, H.; FERNALD, J. **Pair programming improves student retention, confidence, and program quality**. Commun. ACM 49, (8 Aug. 2006), páginas: 90–95.

MELO, C. **Productivity of Agile Teams: An Empirical Evaluation of Factors and Monitoring Process.** 2015. 195 f. Thesis (Doctoral) – Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2015.

MILLS, R.; BECHERT B.; CARRABINE, L. **The Future of Product Development. Computer Aided Engineering**, October 1991.

MOECKEL, A.; AZEVEDO, H. **Aplicação de sistemas computacionais de apoio a gestão do conhecimento no núcleo de pesquisa em Engenharia Simultânea.** Editora novos talentos, v.3, p. 327-337, 2005

MORALES, R.; MCINTOSH, S.; KHOMH, F. **Do Code Review Practices Impact Design Quality? A Case Study of the Qt, VTK, and ITK Projects.** SANER 2015, Montréal, Canada © 2015 IEEE

MUSSNUG, K. J. e HUGHEY A. W. **The trust about teams.** Training for Quality volume 5, nº 1, p. 19-25, 1997.

NASA, *NASA KC1 PROMISE Repository of empirical software.* Disponível em: **<**http://tunedit.org/repo/PROMISE/DefectPrediction**>** Acessado em: 07/09/2015

NBR ISO/IEC 9126 ISO 9126-1:2001, *Software engineering–Product quality – Part 1: Quality model*. International Organization for Standardization, Geneva, Switzerland.

NICOLAESCU, A.; LICHTER, H.; XU, Y. **Evolution of Object Oriented Coupling Metrics: A Sampling of 25 Years of Research.** 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics© IEEE, 2015.

NIKO-NIKO, Niko-niko Calendar. Disponível em: <http://www.geocities.jp/nikonikocalendar/index_en.html>. Acessado em: 10/01/2014

NOSEK, J. **The Case for Collaborative Programming** *Comm. ACM*, Vol. 41, No. 3, 1998, pp. 105–108.

PADBERG, F., MÜELLER, M. **Analyzing the Cost and Benefit of Pair Programming**, in *International Software Metrics Symposium (METRICS) 2003*, pp. 166 – 177 Sydney, Australia, 2003

PADBERG, F., MÜELLER, M. **An Empirical study about the Feelgood Factor in Pair Programming.** In: Proceedings of the 10th International Symposium on Software Metrics METRICS 2004, IEEE Press, 2004.

PADHY, N.; PANIGRAHI, R.; BABOO, S. **A Systematic Literature Review of an Object Oriented Metric:Reusability.** 2015 International Conference on Computational Intelligence & Networks © IEEE, 2015.

PARRISH, A.; SMITH, R.; HALE, D.; HALE, J. **A Field Study of Developer Pairs: Productivity Impacts and Implications.** *Published by the* IEEE Computer Society, pp.76-79. Outubro de 2004.

PHABRICATOR*,* Disponível em: <http://phabricator.org>, Acessado em:18/11/2015

PIMENTEL, C.; AUGUSTO, O. **Engenharia Simultânea e sua aplicação à indústria naval.** Anais do XVIII COPINAVAL. Congresso Panamericano de Engenharia Naval, transporte marítimo e Engenharia Portuária. EPUSP, Brasil, 2003.

PIRES, C.; LOUREIRO, G. **Um ambiente computacional para Engenharia de Sistemas.** Anais do XVI ENCITA. Instituto Tecnológico de Aeronáutica, Brasil, 2010.

PITHON, A. **Projeto Organizacional para a Engenharia Concorrente no Âmbito das Empresas Virtuais.** 2004. 316 f. Tese (Doutorado) - Escola de Engenharia da Universidade do Minho Departamento de Produção e Sistemas. Universidade do Minho, Guinarães, 2004.

PLAUE, C.; LI,Z.; KRAEMER, E. **A Spirit of Camaraderie: The Impact of Pair Programming on Retention** *Department of Computer Science The University of Georgia Athens, GA USA* - IEEE - CSEE&T 2013, San Francisco, CA, USA, pp.209-218.

PLONKA, L.; VAN DER LINDEN, J. **Why developers don't pair more often.** In: Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop, pp 123–125 Conference Location :Zurich, June 2012

PLONKA, L.; SHARP, H.; VAN DER LINDEN, J. **Disengagement in pair programming: Does it matter?** in Proc. 2012 Int'l Conf. Software Eng. (ICSE '12). IEEE Press, 2012, pp. 496–506.

PLONKA, L.; SHARP, H.; VAN DER LINDEN, J.; DITTRICH, Y. **Knowledge transfer in pair programming: An in-depth analysis.** Contents lists available at ScienceDirect journal homepage: <www.elsevier.com/locate/ijhcs>. ©2014 Elsevier Ltd. All rights reserved.

PRASAD, B**. Concurrent engineering fundamentals: integrated product development. v. 2.** ed.1. 1996. New Jersey: Prentice Hall. p.326.

PUTNAM P. TEXEL **Measuring Software Development Status: Do We Really Know Where We Are?** Proceedings of the IEEE SoutheastCon 2015, April 9 - 12, 2015 - Fort Lauderdale, Florida © IEEE, 2015.

ROBINSON, H.; SHARP, H. **XP Culture: Why the twelve practices both are and are not the most significant thing. In:** Agile Development Conference, 2003. Salt Lake City. **Proceedings...** Washington: IEEE Computer Society, 2003. Pp. 12-21

ROOKSBY, J., HUNT, J., WANG, X.: **The theory and practice of randori coding dojos.** In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 251–259. Publisher: Springer International Publishing. Copyright Holder: Springer International Publishing Switzerland. Springer, Heidelberg (2014)

SAMPAIO, S. et al. **A Review of Productivity Factors and Strategies on Software Development**. In Proceedings of the 5th International Conference on Software Engineering Advances (ICSEA'10), 2010, Nice, France, p. 196-204. 101

SANTOS, F.; MOURA, H. **What is wrong with the Software Development?** Research Trends and a new Software Engineering Paradigm. In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA'09), 2009, New York, NY, USA, p. 895-900.

SATO, D. **Uso Eficaz de Métricas em Métodos Ágeis de Desenvolvimento de Software.** Dissertação de mestrado defendida no IME-USP. Orientador: Alfredo Goldman vel Lejbman. Agosto, 2007, São Paulo - Brasil.

SCHWABER, K; BEEDLE, M. **Agile Software Development with Scrum.** Prentice Hall,2001.

SEYAM, M.; MCCRICKARD, S. **Collaborating on mobile app design through pair programming: A practice-oriented approach overview and expert review** Collaboration Technologies and Systems (CTS), 2015 International Conference on Year: 2015 Pages: 124 – 131. © 2015 IEEE

SHIN, J; RUSAKOV,A; MEYER, B. **Concurrent Software Engineering and Robotics Education.** 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015.

SILLITTI, A. ;SUCCI, G. ;VLASENKO, J. **Understanding the Impact of Pair Programming on Developers Attention — A Large, Industrial Case Study,** IEEE Transactions on Software Engineering, junho de 2012, p.1094-1101.

SISON, R. **Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity** Published in: Software Engineering Conference, 2009. APSEC '09. Asia-Pacific Univ., Manila, Philippines. Date of Conference: 1-3 Dec. 2009 Page(s):187 – 193. Publisher: IEEE

SLATEN, K.; WILLIAMS, L.; BERENSON, S., Examining the Impact of Pair Programming on Female Students, NCSU Technical Report, June 17, 2004.

SMITH, J; MENZIES, T. **Should NASA Embrace Agile Processes**. 27th Annual IEEE/NASA *Software* Engineering Workshop, Greenbelt, MD, USA, December 5-6, 2002, Greenbelt Marriott. preprint, West Virginia University,Morgantown, USA.

SUPAKKUL, S.; CHUNG, L. **Representing NFRs and FRs: A Goal-Oriented and Use Case Driven Approach.** Book Title: Software Engineering Research and Applications. Book Subtitle: Second International Conference, SERA 2004, Los Angeles, CA, USA, MAY 5-7, 2004, Revised Selected Papers. pp 29-41. Copyright: Springer-Verlag Berlin Heidelberg 2006

STAA, A. **Programação modular: desenvolvendo programas complexos de forma organizada e segura.** Rio de Janeiro, Campus, 2000. 690p.

STEVENS, R. **Concurrent Engineering Methods and Models for Satellite Concept Design.** The Aerospace Corporation © IEEE, 2015.

STYLECOP, disponível em: <stylecop.codeplex.com>, acessado em:10/09/2015

SUBVERSION*,* disponível em: <subversion.apache.org>, acessado em: 14/07/2015

SWAMIDURAI, R**.;** UMPHRESS**,** D. **The Impact of Static and Dynamic Pairs on Pair Programming.** Published in: Software Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on Date of Conference: June 30 2014-July 2 2014 Page(s): 57 – 63 INSPEC Accession Number:14599783 Conference Location : San Francisco, CA Publisher: IEEE - July 2 2014

SWAMIDURAI, R.; UMPHRESS, D. Inverted pair programming Published in: **SoutheastCon 2015** IEEE 9-12 April 2015 Page(s):1 – 6 INSPEC Accession Number:15240647 Conference Location :Fort Lauderdale, FL., USA, 2015

SWAMIDURAI, R.; KANNAN, U.; DENNIS, B. **Investigating the Impact of Peer Code Review and Pair Programming on Test-Driven Development** Department of Mathematics and Computer Science Alabama State University Montgomery, USA and Dept. of Computer Science and Software Engineering Auburn University Auburn, USA. Published in: IEEE 2014

TERWIESCH, C; LOCH, C.; DE MEYER, A. **Exchanging Preliminary Information in Concurrent Engineering: Alternative Coordination Strategies.** *Organization Science*. , 13(4) , 402-419. Research Collection Lee Kong Chian School Of Business. 2002. **Available at:** http://ink.library.smu.edu.sg/lkcsb_research/3505

TRENDOWICZ, A.; MÜNCH, J. **Factors influencing software development productivity – state of theart and industrial experiences.** Advances in Computers, Elsevier, vol. 77 pp. 185-241, 2009

VALLE, S; BUSTELO, D.V. et al, **Concurrent engineering performance: Incremental versus radical innovation**. IN*: Internal Journal Production Economics* , University of Oviedo, 2009

VANHANEN, J; KORPI, H. **Experiences of Using Pair Programming in an Agile Project**, in *40th Annual Hawaii International Conference on System Sciences (HICSS)2007* Hawaii, 2007, pp. 274b - 274b

VANHANEN, J, LASSENIUS, C; MÄNTYLÄ, M. **Issues and Tactics when Adopting Pair Programming: A Longitudinal Case Study**, in *International Conference on Software Engineering Advances (ICSEA) 2007*, Cap Esterel, French Riviera, France, 2007, p. 70.

WEINBERG, G. **The Psychology of Computer Programming.** New York: Van Nostrand, 1971.

WEINBERG, G. **Software com Qualidade - Volume 3 - Ação Congruente.** Tradução feita por Maria Rosa, revisão técnica feita por Fernão Germano e Paulo Cesar. Do original: Quality Software Management - Volume 3 – Congruent Action. São Paulo, SP. Makron *Books,* 1997.

WEINBERG, G. **The Psychology of Computer Programming.** *Silver Anniversary Edition*. New York: Dorset House Publishing, 1998.

WILLIAMS, L. **Making Software What Really Works, and Why We Believe It**, capítulo: "**Pair Programming**", p.: 311-322. O'reilly Sebastopol, CA–USA, 2011

WILLIAMS, L.; KESSLER, R.; CUNNINGHAM, W.; JEFFRIES, R. **'Strengthening the Case for Pair Programming'.** *IEEE Software,* 2000, p.19–25.

WILLIANS, L; KESSLER,R. **Pair Programming Illuminated**. Massachusetts: Addison-Wesley, 2003.

WILLIANS, L; KESSLER,R **The Effects of 'Pair-Pressure' and 'Pair-Learning' on *Software* Engineering Education.** In: CONF. *SOFTWARE* ENG. EDUCATION (CSEE&T 00), IEEE CS Press, 2000, pp. 59–65.

WILLIAMS, W.; STOUT,M. **Colossal, Scattered, and Chaotic (Planning with a Large Distributed Team**. Proceedings of Agile 2008 Conference (AGILE'2008); 2008, p. 356-361.

WILSON, A. Mob Programming – What's works, what's doesn't - **Agile Processes, in Software Engineering, and Extreme Programming**: proceedings of the 16th International Conference on Agile Software Development, XP 2015, Publisher: Springer International Publishing Switzerland. eBook ISBN: 978-3-319-18612-2, held in Helsinki, Finland, in 25-29 May 2015. Pages: 319-325.

WILSON, A.; WEBER, B. **Product Development in an Unruly Mob.** Agile on the Beach 3&4 September 2015 - 2 DAY TRACK - Track sponsored by Plymouth University Computing - SOFTWARE CRAFTMANSHIP 2015. Disponível em: http://agileonthebeach.com/software-craftmanship-2015/ Acessado em: 07/09/2015

WINNER, R. et al. **The role of concurrent engineering in weapons system acquisition**. E.U.A: Institute for Defense Analyses, 1988. IDA Report R-338.

WINNER, R. **Integrated product/process development in the new attack submarine program: a case study**. 2.ed. s.l. . : Undersecretary of Defense, 2000.

WINNER, R. **The VIRGINIA class submarine program:** a case study**.** Groton, CT : General Dynamics Electric Boat, 2002.

WRAY, S. **How Pair Programming Really Works**. IEEE *Software*, vol. 24, no.6, 2010, pp. 50–55.

WRAY, S. **Responses to "How Pair Programming Really Works"**. IEEE *Software*, March/April 2010, pp. 8–9.

YAN, H. "**Agile concurrent engineering.**" Integrated Manufacturing Systems **Vol.10, nº 2**. MCB University Press, 1999. p 103-112.

ZHANG, T.; SONG, M.; PINEDO, J.; KIM, M. **Interactive code review for systematic changes.** 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. ICSE 2015, Florence, Italy. © IEEE 2015

ZUILL, W. **Mob Programming, All the brilliant people working on the same thing, at the same time, in the same space, and on the same computer.** Publicado em: 14/11/2012 no mobprogramming.org. Disponível em: *http://mobprogramming.org/mob-programming-basics/* Acessado em: 07/09/2015

ZUILL, W. **Mob Programmin: a whole team approach** *at JavaOne.* San Francisco – USA de 28/09 a 02/10 de 2014. Acessado em: 07/09/2015. Disponível em: *<oracleus.activeevents.com/2014/connect/sessionDetail.ww?SESSION_ID=2181>*

ZUILL W. MOB PROGRAMMING, A WHOLE TEAM APPROACH. (Hunter Industries, San Diego California) **Agile on the Beach** 3&4 September 2015 - 2 DAY TRACK - Track sponsored by Plymouth University Computing - SOFTWARE CRAFTMANSHIP 2015. Disponível em: http://agileonthebeach.com/software-craftmanship-2015/ Acessado em: 07/09/2015