THIAGO GANIMI MACHADO

METODOLOGIA ÁGIL – EXTREME PROGRAMMING: PROGRAMAÇÃO EM PAR

ASSIS

2009

METODOLOGIA ÁGIL – EXTREME PROGRAMMING: PROGRAMAÇÃO EM PAR

THIAGO GANIMI MACHADO

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito do Curso de Graduação, analisado
pela seguinte comissão examinadora:

Orientador:_____

Analisador (1):_____

Analisador (2):_____

ASSIS

2009

THIAGO GANIMI MACHADO

METODOLOGIA ÁGIL – EXTREME PROGRAMMING: PROGRAMAÇÃO EM PAR

Trabalho de Conclusão de Curso apresentado ao

Instituto Municipal de Ensino Superior de Assis,

como requisito do Curso de Graduação, analisado

pela seguinte comissão examinadora:

Orientador:_____

Área de Concentração:_____

ASSIS

2009

AGRADECIMENTOS

Ao professor José Augusto Fabri, pela orientação, dicas e sugestões transmitidas durante o trabalho.

Aos professores Alex Sandro Romeo de Souza Poletto e Luiz Ricardo Begosso por ter cedido espaço e programadores para que fosse possível a realização do estudo de caso.

Aos familiares, pelo estímulo e força que me deram.

RESUMO

Esse trabalho teve como objetivo demonstrar os conceitos do Extreme Programming, mas focando uma prática em especial: a programação em par. Além de passar todos os conceitos, foi realizado um estudo de caso para que esses conceitos fossem aplicados para analisar se de fato a programação em par funcionaria, e assim, mostrar para empresas que tenham interesse em usar tal prática que ela pode vir a aumentar a capacidade produtiva da empresa.

No estudo de caso, conseguiu-se levantar dados importantes, o que possibilitou tirar algumas conclusões sobre a programação em par em relação as práticas que foram executadas durante o estudo, como por exemplo a inspeção do código, a troca de idéias e a disseminação do conhecimento.

Palavras-chave: Extreme Programming. Programação em Par.

ABSTRACT

This study aimed to convey the concepts of Extreme Programming, but a practice focusing in particular: pair programming. In addition to passing all the concepts, was performed a case study, so that these concepts were applied to analyze whether in fact the pair programming work, and thus show companies interested in using such a practice, it may well to increase the productive capacity of the company.

In the case study, we were able to get important data, making it possible to draw some conclusions about pair programming for practices that were implemented during the study, such as code inspection, exchange of ideas and dissemination of knowledge.

Keywords: Extreme Programming. Pair Programming.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Hoje em dia, sistemas de software estão em todos os lugares, seja em grandes empresas, em médias empresas e até no armazém do lado de sua casa, porque o software se usado de maneira certa, ajuda e muito na organização de uma empresa. Softwares são usados para vários fins, seja para educação, entretenimento, segurança entre outros.

Para que se tenha um software de alta qualidade é necessário uma engenharia de sofware bem organizada. A engenharia de software é composta pela especificação do software, desenvolvimento e o gerenciamento e evolução destas etapas citadas anteriormente, mas para que isso seja feito de uma forma organizada foram criadas as metodologias de desenvolvimento de projetos.

Ao longo dos anos, foram criadas várias metodologias de desenvolvimento de projetos de software, buscando sempre melhorar a forma de como o software é produzido, pois pesquisas apontam que a grande maioria dos projetos fracassam ou não satisfazem plenamente o cliente.

Por essas razões que o Extreme Programming foi criado. Por ser uma metodologia que é composta por um conjunto de valores, princípios e boas práticas, ele faz com que o trabalho em equipe favoreça, reduzindo assim a taxa de erros em um projeto.

Este trabalho de conclusão de curso propõe-se a mostrar as características do Extreme Programming no desenvolvimento de projetos, mas dando ênfase em uma prática em particular: a programação em par.

2. OBJETIVOS E PROPOSTA

O estudo sobre programação em par tem como objetivo apresentar, por meio de pesquisa e de dados quantitativos, se ela realmente provê um aumento de produtividade nas empresas que a utilizam.

Para atingir o objetivo traçado, será feito um experimento com a programação em par. Este experimento será realizado em uma empresa de desenvolvimento de programas.

Para realizar o experimento, uma mesma tarefa será executada em duas diferentes situações. Na primeira situação, uma tarefa (ainda a ser definida), será atribuída a um grupo de dois programadores, que ocuparão um único computador; na outra situação, a mesma tarefa deverá ser executada por um único programador. No decorrer do experimento serão avaliados vários aspectos como número de vezes em que o programa foi compilado, o tempo em que foi realizada a tarefa, se o objetivo foi alcançado, entre outros resultados. Esses resultados vão gerar dados quantitativos que irão determinar a eficiência ou não da programação em par.

3. EXTREME PROGRAMMING:

O Extreme Programming é uma metodologia de desenvolvimento de software e foi criada no ano de 1996, por Kent Bech, no departamento da montadora de carros Daimler Crysler. O "*XP*" diverge das outras metodologias e pode ser aplicado facilmente em projetos de alto risco devido ao seu dinamismo. Por ser uma metodologia bem organizada e com um conjunto de regras bem definido dentro do modelo, o "*XP*" vem ganhando um grande número de adeptos no mercado e é um modelo de desenvolvimento onde o cliente sempre estará presente, acompanhando a evolução de seu produto e também opinando em algumas situações, fato que aumenta a probabilidade de satisfação ao término do projeto. É portanto, uma

solução que pode minimizar os problemas atuais descritos na literatura sobre o alto índice de projetos que fracassam.

O "*XP*" oferece boas condições de trabalho para que os desenvolvedores tenham eficiência em suas tarefas e devido a sua padronização de projeto, quando um grupo de desenvolvedores termina suas tarefas, eles já sabem exatamente o próximo passo a ser seguido. Toda esta organização que o "XP" prevê, é devido a alguns lemas que devem ser seguidos por seus usuários: Comunicação, Simplicidade, Feedback e Coragem. (mais a frente será detalhado cada um destes itens)

Para que o "*XP*" tenha sucesso, deve ser aplicado em projetos relativamente pequenos, onde sua equipe deva possuir de 2 a 10 integrantes. Estes devem ter conhecimento de todas as fases do desenvolvimento, pois as equipes de desenvolvedores não são fixas, sempre é feito um rodízio, para que a disseminação de conhecimento aconteça sempre e também reforce a idéia do trabalho em grupo. Outro fator para que se obtenha sucesso no "*XP*", é o comprometimento da equipe no projeto, o que gerará um alto índice de produtividade. Acrescente-se também o fato de o cliente estar sempre disponível para tirar dúvidas e tomar algumas decisões em relação ao projeto.

Comentando sobre alguns benefícios do "*XP*" ao mercado, é interessante citar a agilidade no Planejamento (*Planning Games*) que, diferentemente da metodologia tradicional, como por exemplo no Modelo Cascata, onde há contato com o cliente no planejamento e coleta de requisitos e depois ao final do projeto, na entrega do produto final, no "*Planning Games*" não é definida uma especificação completa e formal dos requisitos, e sim, é realizada a produção de sistemas menores ou uma parte do sistema completo, para atender às atuais necessidades do cliente, desta forma, ele nunca ficará na mão quando precisar de uma determinada funcionalidade.

No "*XP*", a comunicação com o cliente é fundamental, pois sempre que preciso, ele deverá estar disponível para tirar as dúvidas, opinar sobre algum requisito e também atribuir prioridades. A linguagem de comunicação com o cliente deve ser feita de acordo com a realidade do cliente. Se for um cliente leigo, não se deve usar muitos termos técnicos, pois ele não entenderá nada e acabará gerando um desconforto

entre desenvolvedor e cliente. Tendo o cliente um bom conhecimento técnico, então não haverá problemas da comunicação ser mais técnica e isso facilitará ainda mais para que o desenvolvedor entenda claramente o que deve ser desenvolvido.

Outra característica importante do "*XP*" é a Programação em Par (*Pair Programming*). A equipe de desenvolvedores é dividida em duplas, cada dupla ocupa um único computador, visando melhorar a qualidade de código.

3.1. CONCEITOS DO EXTREME PROGRAMMING

O "*XP*" vem dando certo nas empresas, pois com ele são criados softwares de melhor qualidade e com um investimento menor, mas para que isso aconteça é obedecida uma série de princípios, valores e práticas que divergem totalmente da forma tradicional de se produzir software.

3.1.1. Princípios

- Auto- semelhança

  Segundo Teles (http://improveit.com.br/xp/principios/auto_semelhanca)

  > O princípio da auto-semelhança sugere que, quando equipes "XP" encontrarem soluções que funcionem em um contexto, também procurem adotá-las em outros, mesmo que em escalas diferentes. Por exemplo, o ritmo básico de desenvolvimento em "XP" é escrever um teste que falha e então fazê-lo funcionar. Esse ritmo opera de forma semelhante em diferentes escalas. Em um trimestre, é listado os temas que quer abordar e então os aborda com histórias. Em uma semana, é listado as histórias que deseja implementar, escreve testes expressando as histórias e então os faz funcionar. Em poucas horas, você lista os testes que sabe que precisa escrever, então escreve um teste, faz funcionar, escreve outro teste, faz os dois funcionarem e assim por diante, até que todos da lista estejam feitos. Esse princípio também se aplica com a adoção de padrões de projeto, por exemplo.

- Benefício Mútuo

  Segundo Teles (http://improveit.com.br/xp/principios/beneficio_mutuo)

  > As práticas do XP são estruturadas de modo a serem mutuamente benéficas para todos os envolvidos em um projeto de software. Programação em par, por exemplo, beneficia os programadores de inúmeras formas. Mas, também beneficia os clientes, porque

> costuma ser raro encontrar bugs em funcionalidades implementadas em par. Gerentes, por sua vez, também se beneficiam porque a programação em par ajuda a disseminar conhecimento na equipe, o que permite que ela supere mais facilmente a ausência de um de seus membros enquanto estiver de férias, por exemplo."
>
> Benefício mútuo é um dos princípios mais importantes do XP e, ao mesmo tempo, um dos mais difíceis de serem adotados. Projetos de software são complexos e normalmente sofrem pressões de tempo e outras que podem levar a equipe a adotar práticas benéficas para uns, mas prejudiciais a outros. É preciso atenção. O bom funcionamento de uma equipe é algo frágil. Práticas que não beneficiem todos os envolvidos são capazes de destruir relacionamentos e criar ainda mais dificuldades nos projetos.

- Diversidade

Essa prática sugere que os desenvolvedores da equipe possuam idéias diferentes, pois na hora de resolver um determinado problema, cada um possa apresentar um ponto de vista diferente e assim escolher a melhor solução para a resolução do problema.

Segundo Teles (http://improveit.com.br/xp/principios/diversidade)

> Práticas como Equipe Integral sugerem que a equipe envolva não só os desenvolvedores, mas também os arquitetos, designers de interação, executivos entre outros, pois opiniões diferentes ajudam a complementar as soluções e torná-las mais ricas.

- Falha

Desenvolvimento de software não é uma atividade simples. Obviamante surgirão alguns impasses durante o desenvolvimento como por exemplo na hora em que surge uma dúvida de como solucionar um determinado problema. Dois membros da equipe de desenvolvimento apresentam soluções distintas para o problema, então surge a dúvida de qual usar, por que não testar as duas soluções então, e a que melhor se encaixar no problema ser adotada?

Segundo Teles (http://improveit.com.br/xp/principios/falha)

> Experimentar diferentes abordagens é importante porque a equipe aprende algo novo. Quando sabemos a melhor solução para um problema, não há muito o que pensar, temos que colocá-la em prática. Mas, quando há dúvida, experimentos podem se revelar melhores que simples discussões. Experimentar diferentes hipóteses e falhar em algumas delas provê novos conhecimentos. Pode

> parecer desperdício, mas quando se trata de aprendizado, freqüentemente a forma mais rápida e rica de aprender é simplesmente tentar algo novo, mesmo que mais tarde tenhamos que voltar atrás e explorar outras alternativas. Em "XP", buscamos feedback concreto. Acreditamos que o código nos ensina o tempo todo, por isso experimentamos e não apenas debatemos sobre possíveis abordagens.

- Humanismo

  Segundo Teles (http://improveit.com.br/xp/principios/humanismo)

> Pessoas desenvolvem software. Metodologias e ferramentas apenas as ajudam a realizar o trabalho. Portanto, é importante compreender a natureza humana para que possamos potencializar o que ela tem de melhor e suprimir o que tem de pior. Em particular, devemos compreender os programadores para que possamos nos aliar a favor e não contra seus instintos.
>
> Programadores gostam de programar e é o que fazem de melhor. Por que forçá-los a escrever documentos, quando não é isso de que gostam, nem é o que sabem fazer de melhor? Não quer dizer que documentação não seja importante ou não deva ser feita. Significa apenas que existem outras formas alternativas de se alcançar o mesmo resultado, porém respeitando a natureza dos desenvolvedores.

- Melhoria

  O princípio de melhoria se resume em sempre se doar ao máximo no projeto, para que a taxa de erros seja mínima. Uma das maneiras de se obter melhorias é ter um feedback frequente com os membros da equipe, o que ajudará a disseminar o conhecimento de forma eficiente e deixará o grupo sincronizado.

- Oportunidade

  Quando surge algum impasse no decorrer de um projeto, o desenvolvedor logo entra em crise. Por que não pensar que este impasse gerou uma oportunidade de aprendizado, pois ocorreu uma situação diferente e querendo ou não ele terá que aprender algo novo, criar alguma solução para resolver o problema.

- Passos de Bebê

  A idéia deste princípio é que sejam feitas pequenas mudanças de cada vez, como por exemplo escrever apenas um trecho de código, testá-lo, caso não funcione, refatorá-lo, testá-lo novamente, então quando estiver funcionando, avançar para o próximo trecho de código e desta forma ir caminhando com o projeto em passos de bebê, porém precisos.

- Qualidade

  Muitas pessoas associam qualidade de software com gastos elevados. Por um lado elas têm razão, porque é óbvio que um produto de qualidade sairá mais caro que um produto de menor qualidade, porém se você produzir um produto de menor qualidade, no fim sairá mais caro, pois o custo que terá para consertá-lo será maior além da insatisfação que irá gerar para o cliente, perda de tempo para correção de defeitos, conflito entre cliente e desenvolvedores entre outros fatores.

  O "*XP*" gera alta qualidade no desenvolvimento de software, através de algumas práticas, tais como:

  - Programação em Par: dois desenvolvedores ocupam o mesmo computador, onde um fica no teclado e o outro inspecionando o código; de tempos em tempos as funções são invertidas. Isto ajuda na disseminação do conhecimento e quanto mais conhecimento existir, maior será a qualidade do software;

  - Desenvolvimento orientado a testes: são testes automatizados que são escritos antes que a implementação da funcionalidade seja realizada. Isso contribui para que se tenha certeza de que a funcionalidade que foi desenvolvida esteja correta.

  - Integração Contínua: as equipes de desenvolvimento integram seus trechos de código com o restante do código em um repositório e várias vezes ao dia, o que ajuda a descobrir eventuais erros.

- Refatoração: as duplas de desenvolvimento revisam seus códigos permanentemente e fazem os ajustes necessários para torná-los mais compreensivos e otimizados.

- Redundância

Na produção de software, sempre lidamos com alguns problemas difíceis de serem resolvidos. No "*XP*", os problemas são tratados de várias formas e em várias etapas, sendo assim, se um erro passa despercebido em uma etapa, na próxima pode ser que ele seja descoberto.

Segundo Teles (http://improveit.com.br/xp/principios/redundancia)

> Não dá para resolver o problema dos defeitos com uma única prática. É excessivamente complexo, com muitas faces e nunca será resolvido completamente. O que você espera alcançar é um número suficientemente pequeno de defeitos para manter a confiança tanto dentro da equipe, quanto com o cliente.
>
> Apesar de redundância poder implicar em desperdícios, tome cuidado para não eliminar redundâncias que servem a propósitos válidos. Ter uma fase de testes após o desenvolvimento estar completo deveria ser redundante. Entretanto, elimine a fase apenas quando se provar que ela se tornou redundante na prática, na medida em que não pega mais nenhum defeito após inúmeras implantações consecutivas.

- Reflexão

O princípio de reflexão diz que não basta apenas as equipes realizarem seu trabalho, é interessante também parar um pouco para refletir sobre a forma como estão trabalhando, e a partir dessa reflexão, ver aonde estavam errando e aprender com estes erros e também encontrar melhores maneiras de se relacionar com o grupo.

Segundo Teles (http://improveit.com.br/xp/principios/reflexao)

> Reflexão pode ser levada longe demais. Desenvolvimento de software tem uma longa tradição de pessoas que se mantêm tão ocupadas pensando sobre desenvolvimento de software que elas não têm sequer tempo para desenvolver software. Reflexão vem depois da ação. Aprendizado é ação refletida. Para maximizar o feedback, reflexões em equipes "XP" são misturadas com ação.

- Responsabilidade Aceita

No “*XP*” a responsabilidade não é simplesmente atribuída e sim, ela é aceita. Quando o gerente lhe passar uma responsabilidade, cabe a você decidir aceitá-la ou não, para aceitá-la deve-se analisar se você possui a capacidade técnica e se também é um momento apropriado para você resolvê- la.

Segundo Teles (http://improveit.com.br/xp/principios/responsabilidade_aceita)

> As práticas refletem responsabilidade aceita, por exemplo, sugerindo que, quem quer que aceita fazer um trabalho também o estime. Da mesma forma, a pessoa responsável por implementar uma história também é responsável pelo design, implementação e teste da mesma.
>
> Com responsabilidade, vem autoridade. Desalinhamentos distorcem a comunicação da equipe. Quando um especialista no processo pode me dizer como devo trabalhar, mas não compartilha aquele trabalho ou é afetado por suas conseqüências, autoridade e responsabilidade estão desalinhadas. Nenhum de nós dois está em uma posição intelectual de ver e usar o feedback de que precisamos para melhorar. Há também um custo emocional de viver com desalinhamentos.

3.1.2. Valores

- Comunicação

A comunicação no “*XP*” é essencial para que um projeto tenha sucesso. Diferentemente das metodologias tradicionais, o cliente é muito envolvido no projeto, pois é ele quem escreve as Histórias que serão passadas para os desenvolvedores trabalharem, dá sugestões e sua opinião também deve ser levada em conta e avaliada. A comunicação entre cliente e equipe de desenvolvimento pode ser feita de várias formas: videoconferências, telefonemas e emails, mas a forma mais eficaz seria o diálogo presencial.

Segundo Teles (http://improveit.com.br/xp/valores/comunicacao)

> Embora existam inúmeras formas de se comunicar idéias, algumas são mais eficazes que outras. Por exemplo, quando duas pessoas estabelecem um diálogo presencial, inúmeros elementos da comunicação colaboram para a compreensão do assunto, tais como gestos, expressões faciais, postura, palavras verbalizadas, tom de voz, emoções, entre outros. Quanto maior a capacidade de compreensão, maiores as chances de evitar problemas como ambigüidades, entendimento equivocados, entre outros. Diálogos são mais eficazes que videoconferências, que são melhores que telefonemas, que são mais expressivos que emails e assim sucessivamente. Conscientes disso, aqueles que trabalham com "XP " priorizam o uso do diálogo presencial, com o objetivo de garantir que todas as partes envolvidas em um projeto tenham a chance de se compreender da melhor maneira possível.

- Coragem

Existem temores que costumam assombrar os participantes de um projeto de software. Beck e Fowler (2001) citam alguns destes temores que exercem influências significativas nos processos de desenvolvimento.

Clientes temem:

- Não obter o que pediram;
- Pedir a coisa errada;
- Pagar demais por muito pouco;
- Não saber o que está acontecendo e
- Fixarem-se em suas primeiras decisões e não serem capazes de reagir a mudanças nos negócios.

Já, os desenvolvedores temem:

- Ser solicitados a fazer mais do que sabem fazer;
- Ser ordenados a fazer coisas que não fazem sentido;
- Ficar defasados tecnicamente;
- Não receber definições claras sobre o que precisa ser feito;

- Sacrificar a qualidade em função do prazo;
- Ter que resolver problemas complicados sem ajuda e
- Não ter tempo suficiente para realizar um bom trabalho.

No “*XP*”, todos estes temores citados acima podem vir a acontecer e a equipe de desenvolvimento buscará várias formas para lidar com os problemas de maneira corajosa. Para Teles (2005), “ter coragem no “*XP*” significa ter confiança nos mecanismos de segurança utilizados para proteger o projeto”, isto é, quando os problemas ocorrerem e de fato ocorrerão, os princípios e práticas do “*XP*” serão utilizados para que possam ajudar a reduzir ou eliminar as consequências destes problemas.

- Feedback

No modelo “XP”, feedback é a comunicação entre o cliente e a equipe de desenvolvimento e também entre os própios membros da empresa. Quanto mais feedback ocorrer, a chance de um projeto ter sucesso é bem maior, pois o cliente estará sempre acompanhando de perto o desenrolar do projeto e poderá dar suas sugestões em relação a ele.

Segundo Teles (2005)

> O Extreme Programming é organizado em ciclos curtos de feedback que possibilitem aos usuários solicitar funcionalidades e aprender sobre elas através de software funcionando em prazos curtos. Esse processo envolve a priorização de poucas funcionalidades a serem implementadas de cada vez e a simplificação das mesmas, na medida do possível. O objetivo se torna apresentar a funcionalidade ao usuário rapidamente, de modo que ele possa, cedo, detectar eventuais falhas, quando tende a ser mais barato corrigí-las.

- Respeito

Para ter boas condições de trabalho em um ambiente é necessário este valor e sem ele, dificilmente os outros valores irão funcionar. Portanto, saber a hora certa de falar, ouvir, quando estiver errado assumir o erro e respeitar o colega de profissão são valores fundamentais para que um projeto de software seja bem sucedido.

- Simplicidade

  O “*XP*” utiliza o conceito de simplicidade para o desenvolvimento de projetos, pois ele prega que a equipe se concentre apenas em desenvolver aquilo que tem prioridade no momento e não desenvolver aquilo que poderia vir a ser necessário, pois o cliente pode facilmente querer mudar de idéia e tudo aquilo que foi feito a mais terá sido em vão. “Se você mantiver o sistema suficientemente simples o tempo todo, qualquer coisa que você coloque nele será inserido facilmente e na menor quantidade de lugares possível (JEFFRIES, ANDERSON, pg 76, tradução nossa).”

3.1.3. Práticas

- Ambiente Informativo

  O ambiente de trabalho da equipe de desenvolvimento deve ser o reflexo do projeto em que ela esteja trabalhando. Para que isto aconteça, as equipes “*XP*” utilizam diversos instrumentos, tais como:

  - Cartões de histórias colocados em um mural na parede;
  - Quadro branco, para serem postadas algumas informações;
  - Post it sobre as paredes;
  - Flip chart.

  Essa prática permite que a equipe ganhe tempo, pois as informações mais relevantes sobre o projeto em que estão trabalhando, estarão sempre visíveis nas paredes. Essas informações devem ser sempre atualizadas, para que a equipe também esteja sempre atualizada.

- Build de Dez Minutos

  Build em poucas palavras, seria o processo de compilar, montar e empacotar o programa. Esta prática, prega que os Builds levem em média de dez minutos para serem executados, pois se forem muito demorados, a tendência é que a equipe o execute com menos frequência e isto pode acarretar no

acúmulo de erros, ficando mais complicado para solucionar o problema depois. Logo, esta prática dá oportunidade para a equipe receber feedback sobre o funcionamento do sistema de forma mais rápida.

- Ciclo Semanal

  Como no "*XP* " os projetos são desenvolvidos de modo iterativo e incremental, isto é, por etapas, pelo menos uma vez por semana, clientes e desenvolvedores se reúnem para definir o que será implementado e testado na semana. Após uma semana, o cliente recebe o que foi feito, utiliza e avalia o que foi produzido. A partir desta avaliação, ele se reúne novamente com a equipe e estabelece novas prioridades e assim este ciclo fica se repetindo.

- Ciclo Trimestral

  Os planejamentos de projetos no "*XP*" são divididos em trimestres. No ínicio de cada trimestre o cliente se reúne com a equipe de desenvolvimento para discutir o tema que será abordado nos próximos três meses. Temas são "conjuntos de funcionalidades que solucionam as necessidades de um ou mais processos de negócios da organização". (Teles, http://improveit.com.br/xp/praticas/ciclo_trimestral)

- Desenvolvimento Orientado a Testes

  Hoje em dia, falhas de software são muito comuns e estão presentes até nas grandes empresas de desenvolvimento. Dependendo da falha, pode ser uma coisa fácil de se contornar e isso não gerará um prejuízo muito grande, mas se estivermos falando de um projeto grande e ocorrer uma falha grave, isto pode significar um enorme prejuízo financeiro. Para tentar reduzir ao máximo as falhas de software, o "*XP*" usa a prática do Desenvolvimento Orientado a Testes, que nada mais é que "criar uma base de testes automatizados que possam ser executados toda vez que um novo fragmento de código é adicionado ao sistema. Embora isso não impeça a ocorrência de erros, representa um instrumento útil para detectá-los rapidamente, o que agiliza a

correção e evita que eventuais bugs se acumulem ao longo do tempo.” (TELES, 2005)

- Folga

Esta prática do “*XP*” prega que os funcionários da empresa trabalhem no máximo quarenta horas semanais, isto é, oito horas por dia, trabalhando mais que isso, entende-se que ocorrerá uma queda de rendimento dos funcionários e ele não renderá o que poderia render se estivesse cem por cento descansado físicamente e mentalmente.

Segundo (POPPENDIECK & POPPENDIECK, 2003, p.81, tradução nossa)

> Utilização plena não provê nenhum valor para a cadeia de valor como um todo; de fato, normalmente faz mais mal que bem. Assim como uma rodovia não consegue prover serviços aceitáveis sem alguma folga na sua capacidade, você provavelmente não estará provendo aos seus clientes o nível mais elevado de serviço se não tiver alguma folga em sua organização.

- Histórias

Histórias nada mais são que o cliente escrever em cartões as funcionalidades que ele deseja que seja implementada pelos desenvolvedores. Elas devem ser escritas de maneira simples e compreensível e não serem muito longas, pois elas serão repassadas para os desenvolvedores trabalharem em cima daquilo que está escrito.

Cliente:

- Define as histórias;
- Decide qual o valor de negócio de cada história e
- Decide que histórias serão construídas no release.

Desenvolvedores:

- Estimam o tempo necessário para construção de cada história;
- Advertem o cliente sobre riscos técnicos significativos e

- Medem o progresso da equipe para fornecer um orçamento geral para o cliente (BECK & FOWLER, 2001, p.40, tradução nossa).

- Integração Contínua

  Integração contínua é os pares trabalharem de forma isolada, porém, integrarem o que produzem com a versão mais recente do código de produção, diversas vezes ao dia. Isto é, os pares se sincronizam com frequência à medida que terminam pequenas atividades de codificação.(BECK, 2000).

  Por a integração ser contínua, isso facilita para que erros sejam identificados sempre, desta forma o código pode ser corrigido rapidamente. Imagine o problema que seria caso um projeto ficasse uma semana sem ser feita a integração, a quantidade de erros que se acumulariam, sem contar que a equipe estaria totalmente fora de sincronia.

- Programação em Par

  Dois programadores ocupam o mesmo computador, onde um fica no teclado e o outro inspecionando o código; de tempos em tempos, as funções são invertidas. Isto ajuda na disseminação do conhecimento e quanto mais conhecimento existir, maior será a qualidade do software. Esta prática é bem interessante também pelo fato de que dificilmente o programador irá relaxar e negligenciar o serviço, checar emails, entrar em chats, pois sempre terá outro programador ao seu lado, fato que inibirá estas ações.

- Sentar-se junto

  Esta prática do "*XP*" fala que as equipes de projeto devem sempre se sentarem juntas, de preferência em uma sala aberta, onde todos possam se comunicar facilmente para tirar suas dúvidas e discutir algumas situações.

- Código Coletivo

  No "XP" todos membros da equipe têm acesso ao código fonte do projeto por completo e os pares de desenvolvimento se revezam no desenvolvimento deste projeto, ora estão desenvolvendo uma funcionalidade, ora estão

desenvolvendo outra funcionalidade. Pelo fato de todos terem acesso ao código, o projeto anda de forma mais rápida e se ganha um tempo valioso.

- Reunião em pé

Conhecida também como Stand Up Meeting, essa prática serve para que a equipe de projeto realize uma rápida reunião diária a fim de serem discutidos os resultados obtidos no dia anterior para que se possa priorizar as atividades do dia que se inicia.

Segundo Teles (2004, p.88)

> No fim do stand up meeting, cada membro da equipe sabe o que deverá fazer ao longo do dia. É importante notar que a decisão sobre o que fazer ao longo do dia não é tomada por uma única pessoa. Essa decisão é tomada em equipe. Isso faz sentido, porque quando todos se reúnem, é possível ter uma visão de todo o desenvolvimento e não apenas de uma parte. Desta forma, é factível decidir com mais eficácia quais são as prioridades do dia.

- Refatoração

Refatoração é o processo de fazer mudanças em um código existente e funcional sem alterar seu comportamento externo. Em outras palavras, alterar como ele faz, mas não o que ele faz. O objetivo é aprimorar a estrutura interna (ASTELS, 2003, pg.15, tradução nossa).

As duplas de desenvolvimento revisam seus códigos permanentemente e fazem os ajustes necessários para torná-los mais compreensivos e otimizados.

4. Programação em Par

Para Willians e Kessler (2003, p.3, tradução nossa), programação em par é “um estilo de programação na qual dois programadores trabalham lado a lado em um computador, continuamente, colaborando no mesmo design, algorítmo, código e teste.”

Isto é uma coisa totalmente positiva, pois enquanto um programador está codificando, o parceiro ao lado esta inspecionando o código. É muito mais fácil para uma pessoa que não está programando identificar um erro, porque às vezes o programador está tão concentrado no que faz que um erro pode passar despercebido.

“Vários experimentos confirmam que o olho tem uma tendência de ver o que ele espera ver (WEINBERG, 1971, p.162, tradução nossa)”, desta forma, a programação em par se mostra eficente em relação a inspeção de código.

Segundo Williams & Kessler (2003, pg.27, tradução nossa) “a teoria sobre por que as inspeções de código são eficazes se baseia no conhecimento proeminente de que quanto mais cedo um defeito é encontrado em um produto, mais barato é o seu conserto.”

Revisões em pares representam uma das formas mais eficazes para encontrar defeitos em software. As evidências que dão suporte às revisões em pares remontam a mais de duas décadas e ninguém mais questiona seus benefícios (EMAM, 2003 *apud* TELES, 2005).

Segundo Teles (http://improveit.com.br/xp/praticas/programacao_par)

> Quem trabalha continuamente com programação em par se habitua a corrigir e ter seu trabalho corrigido dezenas de vezes ao dia. A incidência de erros identificados pelo colega costuma ser tão elevada que surpreende quem não está acostumado ao uso da técnica. Depois de um dia ou dois experimentando a programação em par, muitas pessoas se perguntam como alguém as permite trabalhar sozinhas. É o tipo de coisa que não é intuitivo, mas acontece e só é claramente perceptível quando se experimenta a técnica.

Porém, não são todas as pessoas que se adaptam a esta técnica. Existem programadores que não admitem que outro programador dê opinião sobre como resolver um problema ou dizer que algum trecho de código está errado, pois são

extremamente individualistas e têm dificuldade para se trabalhar em equipe. Em uma equipe que trabalha com o "*XP*" não se pode ter uma pessoa assim, individualismo vai contra os valores e princípios do "*XP*".

Outro aspecto importante da programação em par é que duas cabeças pensam melhor que uma. Pode-se chegar mais rápido na resolução de um problema, como pode-se também discutir idéias, consegue-se explorar mais alternativas e decidir qual é a mais simples e eficaz para resolver um determinado problema. Geralmente, se um desenvolvedor está sozinho e tem um problema para ser resolvido, a primeira solução que ele achar será a adotada, pois não terá outra pessoa ao lado para discutir a idéia.

Segundo Williams, Kessler (2000, pg.20, tradução nossa)

> Os pares consideram muito mais soluções possíveis para um problema e convergem mais rapidamente para a solução que será implementada. De acordo com os pesquisadores Nick Flor e Edwin Hutchins, o feedback, debate e troca de idéias entre parceiros, reduzem significativamente a probabilidade de proceder com um design ruim.

A programação em par é muito útil também para a troca de conhecimentos na equipe de desenvolvimento. "O conhecimento é passado constantemente entre os parceiros, desde dicas de utilização das ferramentas até as regras da linguagem de programação (...) os pares se revezam no papel de professor e aluno (WILLIAMS & KESSLER, 2003, p.29, tradução nossa).

Uma empresa contrata um novo desenvolvedor, este não tem nenhum conhecimento da ferramenta usada pela empresa, sentando ao lado de um programador, lhe será explicado todo o funcionamento da ferramenta, ele poderá tirar dúvidas e ver a forma como trabalha o desenvolvedor.

"A presença imediata de um colega que tenha conhecimentos importantes sobre o problema em questão, torna o processo de aprendizado imeditado (TELES, 2005)".

Quando um dos membros do par tem menos experiência que o outro, é interessante que ele retire o máximo de conhecimento do parceiro, através de perguntas e até mesmo assumindo o teclado e sendo apenas guiado, pois a melhor maneira de aprender algo é fazendo.

A programação em par também eleva a robustez da equipe, permitindo que ela supere melhor a perda ou a adição de um novo membro na equipe.

Segundo Williams, Kessler (2003, pg.41, tradução nossa) “com a programação em par, o risco do projeto associado à perda de um programador chave é reduzido porque existem várias pessoas familiarizadas com cada parte do sistema”. Além disso, novos desenvolvedores podem começar a contribuir mais cedo.

Segundo Williams, Kessler (2003, pg.42, tradução nossa)

> Tradicionalmente, pessoas novas em uma organização são apresentadas a diferentes partes de um sistema por uma pessoa experiente da equipe. Este tempo dedicado ao treinamento custa diversas horas valiosas do membro experiente. Durante estas horas, nem a pessoa nova, nem a experiente estão contribuindo para completar o projeto.

Porém, ao término desse treinamento, a equipe poderá contar com mais um desenvolvedor para poder completar o projeto.

Falando um pouco agora sobre a pressão do par, “se um desenvolvedor trabalha oito horas por dia, engana-se quem pensa que ele realmente produza durante essas oito horas. De modo geral, o aproveitamento de um programador trabalhando sozinho raramente chega sequer a 50% do tempo que passa na frente do computador (TELES, http://improveit.com.br/xp/praticas/programacao_par)”.

Isto ocorre por várias razões, pois programar é complexo e é uma atividade que exige atenção da pessoa. Por ser uma atividade complexa, chega uma hora em que o desenvolvedor cansa e vai se distrair um pouco olhando seus emails, conversando com o colega ao lado entre outras coisas. É neste momento que entra a pressão do par. Nestas horas em que bate o desânimo no programador o seu parceiro irá incentivá-lo a dar continuidade ao trabalho, pois a partir do momento em que um para o outro será prejudicado. Em relação a consultar os emails, a pessoa pensará duas vezes, pois não é uma situação confortável consultar emails com uma pessoa do lado.

Segundo Teles (http://improveit.com.br/xp/praticas/programacao_par) ”não gostamos de decepcionar nossos colegas, portanto, quando programamos em par, procuramos manter o maior foco possível na atividade para não atrapalhar ninguém. A responsabilidade é maior e isso faz com que o foco também seja”.

A principal característica da programação em par é a disseminação do conhecimento, porque além de se trabalhar em dupla, os pares não são fixos, de

tempos em tempos eles são alterados, isso facilita para a troca de informações e conhecimento e também os desenvolvedores não trabalham em projetos fixos, hoje ele está desenvolvendo uma funcionalidade, amanhã ele pode estar trabalhando em outra funcionalidade e seu parceiro pode ser outro também, isto faz com que o desenvolvedor tenha conhecimento de todo o projeto e não apenas de uma determinada funcionalidade dele.

Segundo Beck (2000 apud Teles 2005)

> Programação em par funciona para o "*XP*" porque encoraja a comunicação. Gosto de pensar na analogia com um copo d'água. Quando uma informação importante é aprendida por alguém na equipe, é como se colocássemos uma gota de tintura na água. Já que os pares se revezam o tempo todo, a informação se difunde rapidamente através da equipe, da mesma forma que a tintura se espalha através da água. Ao contrário da tintura, entretanto, a informação vai ficando mais rica e mais intensa à medida que se espalha e é enriquecida pela experiência e idéias de todas as pessoas da equipe

A programação em par é uma das práticas do "*XP*" mais polêmicas, pois muitas pessoas entendem que não é necessário duas pessoas fazerem o trabalho de uma, acham que só haverá mais gastos, porém não é exatamente isso o que ocorre.

Segundo Teles (2005)

> "Willians e Kessler (2003) executaram experimentos, com resultados estatisticamente significativos, demonstrando que a programação em par eleva o número de programador-hora em apenas 15 por cento. Entretanto conseguiram "validar estatisticamente os relatos que alegam que a programação em par é um meio acessível de produzir software de mais elevada qualidade (WILLIAMS & KESSLER, 2003, p.9, tradução nossa). Ou seja, apesar da ligeira elevação no consumo de recursos, os resultados mostraram que os desenvolvedores que trabalharam em par geraram um número significativamente menor de defeitos, produziram menos código para solucionar o mesmo problema e tiveram resultados mais consistentes (WILLIAMS & KESSLER, 2000)"
>
> Trabalhando em sintonia, os pares completaram suas atribuições de quarenta a cinquenta por cento mais rápido. No mercado atual, obter um produto de qualidade tão rapidamente quanto possível é uma vantagem competitiva que pode até significar a prórpia sobrevivência. Consertar defeitos depois de lançar os produtos para os clientes pode custar muito mais que encontrá-los e consertá-los durante o desenvolvimento. Os benefícios de ter um produto lançado mais rapidamente, com gastos menores de manutenção e que

> melhora a satisfação dos clientes supera qualquer aumento na quantidade de programador-hora que possa resultar dos pares (WILLIAMS, KESSLER , 2000, p.22, tradução nossa).

4.1. ESTUDO DE CASO

Neste capítulo será feito um estudo de caso com a programação em par, de acordo com o que foi sugerido nos objetivos e proposta do trabalho.

4.1.1. Protocolo do Estudo de Caso

A- Estabelecimento do caso

Foi realizado um estudo de caso com a programação em par, onde duas tarefas de complexidades distintas foram executadas em duas diferentes situações. Na primeira situação, uma tarefa foi atribuída a um grupo de dois programadores, que ocuparam um único computador; na outra situação, a outra tarefa foi executada por um único programador. No decorrer do experimento, foram avaliados vários aspectos assim como o tempo total em que foi realizada a tarefa, se o objetivo foi alcançado, entre outros resultados. Esses resultados geraram dados quantitativos que determinaram a eficiência ou não da programação em par.

B- Local e tempo para a realização do estudo de caso

O estudo de caso foi realizado na faculdade FEMA (Fundação Educacional do Município de Assis), mais especificamente no CEPEIN (Centro de Pesquisa em Informática) e levou um tempo total de 3 horas, começando às 14h e terminando às 17h.

C- Participantes e papéis

O estudo de caso envolveu dois programadores do CEPEIN, que tiveram como objetivo desenvolver duas tarefas, onde a primeira delas, um dos programadores desenvolveu individualmente e em seguida se juntou ao outro programador formando o par para desenvolver a segunda tarefa.

D- Pré-questionário

Antes das tarefas terem sido executadas, foi aplicado um pré-questionário, visando colher alguns dados dos programadores para reconhecimento dos perfis para a formação do par. O questionário elaborado possui nove questões, entre elas questões de múltipla escolha e questões dissertativas. Segue abaixo o pré-questionário:

1) Você é formado?

( ) Sim ( ) Não

2) Qual o curso que você fez ou está fazendo ?

( ) BCC ( ) TPD ( ) Nenhum

( ) Outros: Informe quais_____

3) Quais as linguagens de programação que você já aprendeu no decorrer do curso?

( ) C/C++ ( ) Visual Basic ( ) Delphi ( ) Java

( ) Outros: Informe quais:_____

4) Com quais linguagens de programação você trabalha?

( ) C/C++ ( ) Visual Basic ( ) Delphi ( ) Java

( ) Outros: Informe quais:_____

5) Qual a sua carga horária de trabalho diária?

( ) 4 horas ( ) 6 horas ( ) 8 horas ( ) Outra: Informe:_____

6) De que projeto você faz parte na empresa em que trabalha?

7) Qual sua função nesse projeto?

8) Você costuma interromper muitas vezes seu trabalho para conversar, checar e-mails?

( ) Nunca ( ) Às vezes ( ) Sempre

9) Classifique a funcionalidade que será desenvolvida:

( ) Fácil ( ) Médio ( ) Difícil

E- Execução do estudo de caso

O estudo de caso foi realizado em duas etapas distintas: na primeira etapa foi passada uma tarefa para um dos programadores, que era realizar a manutenção no cadastro de domicílios. A manutenção deveria permitir a exclusão de um registro quando não houvesse dependências em outras tabelas. Já na segunda etapa, outra tarefa foi executada, mas com os dois programadores trabalhando em par. A tarefa era ajustar a interface dos botões na tela e adequar a resolução, pois em algumas resoluções a imagem estava sendo cortada. Vale lembrar que as tarefas que foram designadas a fazer, foram alterações em funcionalidades de um sistema que é desenvolvido no própio CEPEIN; é um sistema de Ação Social que é feito na linguagem de programação Visual Basic. Essas tarefas foram escolhidas pelos próprios funcionários do CEPEIN, pois não podia ser executada qualquer tarefa, pois iria atrapalhar a produtividade dos desenvolvedores. Então essas tarefas faziam parte do trabalho que eles tinham que realizar no dia.

F- Pós-questionário

Após o pré-questionário ser aplicado, as tarefas terem sido definidas e os grupos formados, começaram a ser executadas as tarefas por ambos os grupos e ao término das tarefas, foi passado um pós-questionário para o par. Este questionário visou colher dados em relação à experiência que tiveram trabalhando em par e também os dados da tarefa que realizaram, como por exemplo, o tempo total para a realização da tarefa para poder ser comparado com os dados do programador que realizou a tarefa individualmente e assim estabelecer um resultado. Segue abaixo o

pós- questionário, lembrando que nele, algumas perguntas foram baseadas na teoria da programação em par. Então cada uma dessas questões será seguida por justificativas, o porquê e em que foram baseadas.

1) O que achou dos conceitos em geral da Programação em Par?

( ) Muito Ruim ( ) Ruim ( ) Regular ( ) Bom ( ) Muito Bom

2) Você se sentiu à vontade trabalhando em par?

( ) Sim ( ) Não

Um dos problemas que se tem programando em par é o ego do programador. Caso ele seja uma pessoa muito individualista, fica difícil de se trabalhar.

Segundo Teles (http://improveit.com.br/xp/praticas/programacao_par)

> Weinberg criou o termo egoless programming, ou seja, programação sem ego. É essencial que o desenvolvedor não veja o código como uma extensão de si próprio. Ele não pode achar que um erro no código significa uma falha sua como pessoa. O código é apenas uma obra que tem vida própria e não representa o caráter e a personalidade de quem o criou.

3) Em relação a sua produtividade trabalhando em par:

( ) Piorou ( ) Continuou mesma coisa ( ) Melhorou

4) Houve conflito de idéias entre você e seu par?

( ) Sim ( ) Não

5) Caso tenha tido um conflito de idéias, chegou-se a um consenso de qual seria a melhor opção a seguir?

( ) Sim ( ) Não

Segundo (WILLIAMS, KESSLER, 2000, p.20, tradução nossa)

> Os pares consideram muito mais soluções possíveis para um problema e convergem mais rapidamente para a solução que será implementada, o feedback, debate e troca de idéias entre os parceiros reduzem significativamente a probabilidade de proceder com um design ruim.

6) Em relação à inspeção de código, localização de erros e bugs no programa:

( ) Demorou-se mais para encontrar o erro

( ) O par não fez diferença por ter encontrado o erro

( ) Encontrou-se mais rápido o erro

Segundo (EMAM, 2003, p.12, tradução nossa)

> Revisões em pares representam uma das formas mais eficazes para encontrar defeitos em software. As evidências que dão suporte às revisões em pares remontam a mais de duas décadas e ninguém mais questiona seus benefícios."

7) Em relação à interrupção do trabalho para conversas em chats e checar e-mails, você ficou inibido(a) pelo seu par?

( ) Sim ( ) Não

Esta questão se baseia no conceito "Pressão do Par".

Segundo (EMAM, 2003, p.12, tradução nossa)

> Ter um parceiro trabalhando com você durante a maior parte do dia minimiza distrações e interrupções. Um desenvolvedor dificilmente irá bater papo sobre assuntos de lazer ao telefone enquanto uma pessoa estiver sentada ao seu lado esperando que termine. Da mesma forma, alguém dificilmente irá interromper duas pessoas no meio de uma discussão. Portanto, a quantidade de tempo não produtivo que é recuperado pode levar a um aumento de produtividade.

8) Por estar programando em par, você aprendeu com seu companheiro e também ele conseguiu aprender com você?

( ) Sim ( ) Não

“Durante o trabalho dos pares, o conhecimento é passado constantemente entre os parceiros, desde dicas de utilização das ferramentas até regras da linguagem de programação, os pares se revezam no papel de professor e aluno.” (WILLIAMS E KESSLER, 2003, p.29, tradução nossa)

9) Qual foi o tempo total para a realização da tarefa?

10) Muitos profissionais da área não concordam com a programação em par, alegando que ao se usarem duas pessoas para fazer o trabalho de uma, logo se gastaria o dobro para a remuneração dos profissionais. Você concorda com esta alegação ou tem um pensamento diferente? Justifique sua resposta.

Programação em par é uma das práticas mais conhecidas e mais polêmicas utilizadas pelos que adotam o “Extreme Programming”.(...) *À primeira vista*, a programação em par parece ser uma prática fadada ao fracasso e ao desperdício (...) temos a impressão de que ela irá consumir mais recursos ou irá elevar o tempo de desenvolvimento. (TELES, http://improveit.com.br/xp/praticas/programacao_par)

4.1.2. Dados gerados com o Estudo de Caso

Nesta sessão serão apresentados todos os dados que foram colhidos, incluindo as visões dos programadores que participaram do estudo de caso e suas opiniões em relação à programação em par.

Logo abaixo, será apresentado o pré-questionário respondido pelos dois programadores para que se consiga fazer uma análise mais elaborada:

Programador 1:

1) Você é formado?

( ) Sim (x) Não

2) Qual o curso que você fez ou está fazendo ?

(x) BCC

( ) TPD

( ) Nenhum

( ) Outros: Informe quais

3) Quais as linguagens de programação que você já aprendeu no decorrer do curso?

(x) C/C++ (x) Visual Basic (x) Delphi (x) Java ( ) Outros: Informe quais

4) Com quais linguagens de programação você trabalha?

( ) C/C++ (x) Visual Basic ( ) Delphi ( ) Java ( ) Outros: Informe quais:

5) Qual a sua carga horária de trabalho diára?

( ) 4 horas (x) 6 horas ( ) 8 horas ( ) Outra: Informe:_____

6) De que projeto você faz parte na empresa em que trabalha?

R: Sistema de Ação Social

7) Qual sua função nesse projeto?

R: Programador

8) Você costuma interromper muitas vezes seu trabalho para conversar, checar e-mails?

( ) Nunca (x) Àsvezes ( ) Sempre

9) Classifique a funcionalidade que será desenvolvida:

(x) Fácil ( ) Médio ( ) Difícil

Figura 1 – Quadro com o pré-questionário do *programador 1*.

Programador 2:

1) Você é formado?

( ) Sim (x) Não

2) Qual o curso que você fez ou está fazendo ?

(x) BCC

( ) TPD

( ) Nenhum

( ) Outros: Informe quais:

3) Quais as linguagens de programação que você já aprendeu no decorrer do curso?

(x) C/C++ ( ) Visual Basic ( ) Delphi ( ) Java ( ) Outros: Informe quais:

4) Com quais linguagens de programação você trabalha?

(x) C/C++ ( ) Visual Basic ( ) Delphi ( ) Java ( ) Outros: Informe quais:

5) Qual a sua carga horária de trabalho diára?

( ) 4 horas ( ) 6 horas (x) 8 horas ( ) Outra: Informe:_____

6) De que projeto você faz parte na empresa em que trabalha?

R: Desenvolvimento de sites

7) Qual sua função nesse projeto?

R: Programador

8) Você costuma interromper muitas vezes seu trabalho para conversar, checar e-mails?

( ) Nunca (x) Àsvezes ( ) Sempre

9) Classifique a funcionalidade que será desenvolvida:

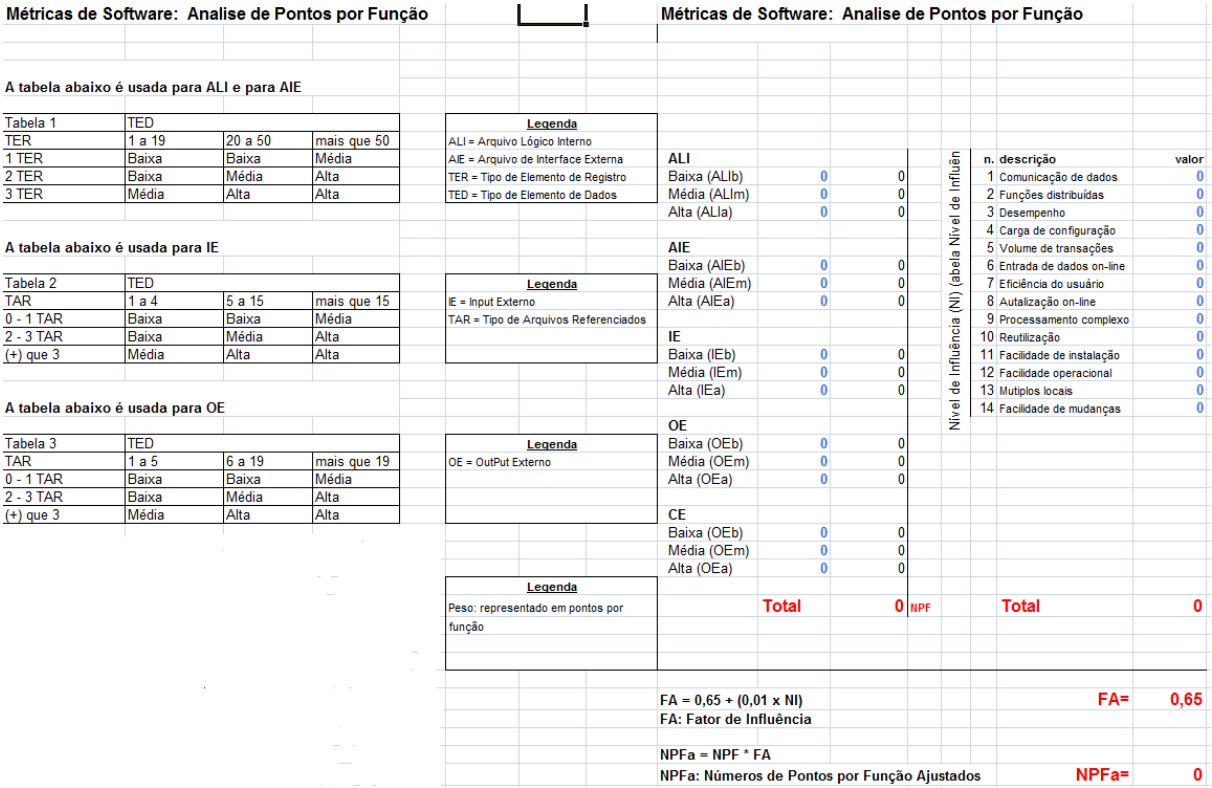
(x) Fácil ( ) Médio ( ) Difícil

Figura 2 – Quadro com o pré-questionário do *programador 2*.

A primeira tarefa foi realizada pelo *programador 1*, por ele já ter experiência com a linguagem Visual Basic e trabalhar com ela normalmente no dia a dia. O programador levou um tempo total de 45 minutos para realizar a tarefa e ao término dela foi conferido para ver se foi atendido o que era para ser feito e estava tudo correto. Já a segunda tarefa foi realizada em par, pelo *programador 1* e *programador 2*, sendo que o *programador 1* assumiu o teclado e o *programador 2* ficou ao seu lado para realizar a inspeção do código, e o tempo total para a realização dessa tarefa foi de 35 minutos.

Para se estabelecer uma comparação entre a primeira tarefa, que foi realizada individualmente e a segunda tarefa, que foi realizada em par, foram contados os pontos por função de cada tarefa, que nada mais é que uma técnica para a medição de projetos de desenvolvimento de software para se estabelecer uma medida de tamanho em pontos por função. Sendo assim, fica-se mais fácil realizar a comparação entre as tarefas.

Para se contar os pontos por função de um projeto usa-se a interface do projeto e também o DER (Diagrama de Entidade e Relacionamento) junto com seus relacionamentos. Usa-se uma tabela (vide figura 3) para fazer esta contagem. Conta-se o número de campos de cada tabela (TEDs), os TERs que são os tipos de elementos de registro, os AIEs, que são os arquivos de interface externa, os ALIs, que são os arquivos lógicos internos e os TARs que são os tipos de arquivos referenciados. Cada um desses itens tem um determinado peso na hora da contagem e ao somar todos se obtém a quantidade de pontos por função.



Métricas de Software: Analise de Pontos por Função

A tabela abaixo é usada para ALI e para AIE

Tabela 1	TED		
TER	1 a 19	20 a 50	mais que 50
1 TER	Baixa	Baixa	Média
2 TER	Baixa	Média	Alta
3 TER	Média	Alta	Alta

Legenda
ALI = Arquivo Lógico Interno
AIE = Arquivo de Interface Externa
TER = Tipo de Elemento de Registro
TED = Tipo de Elemento de Dados

A tabela abaixo é usada para IE

Tabela 2	TED		
TAR	1 a 4	5 a 15	mais que 15
0 - 1 TAR	Baixa	Baixa	Média
2 - 3 TAR	Baixa	Média	Alta
(+) que 3	Média	Alta	Alta

Legenda
IE = Input Externo
TAR = Tipo de Arquivos Referenciados

A tabela abaixo é usada para OE

Tabela 3	TED		
TAR	1 a 5	6 a 19	mais que 19
0 - 1 TAR	Baixa	Baixa	Média
2 - 3 TAR	Baixa	Média	Alta
(+) que 3	Média	Alta	Alta

Legenda
OE = OutPut Externo

Legenda
Peso: representado em pontos por função

Métricas de Software: Analise de Pontos por Função

ALI			
Baixa (ALIb)	0	0	
Média (ALIm)	0	0	
Alta (ALIa)	0	0	
AIE			
Baixa (AIEb)	0	0	
Média (AIEm)	0	0	
Alta (AIEa)	0	0	
IE			
Baixa (IEb)	0	0	
Média (IEm)	0	0	
Alta (IEa)	0	0	
OE			
Baixa (OEb)	0	0	
Média (OEm)	0	0	
Alta (OEa)	0	0	
CE			
Baixa (OEb)	0	0	
Média (OEm)	0	0	
Alta (OEa)	0	0	
	Total	0	NPF

Nível de Influência (NI) (abela Nível de Influên

n.	descrição	valor
1	Comunicação de dados	0
2	Funções distribuídas	0
3	Desempenho	0
4	Carga de configuração	0
5	Volume de transações	0
6	Entrada de dados on-line	0
7	Eficiência do usuário	0
8	Autalização on-line	0
9	Processamento complexo	0
10	Reutilização	0
11	Facilidade de instalação	0
12	Facilidade operacional	0
13	Mutiplos locais	0
14	Facilidade de mudanças	0
	Total	0

FA = 0,65 + (0,01 x NI)
FA: Fator de Influência
FA= 0,65

NPFa = NPF * FA
NPFa: Números de Pontos por Função Ajustados
NPFa= 0

Figura 3 – Tabela para cálculo de pontos por função

Abaixo segue a entidade relacionamento da tarefa 1:

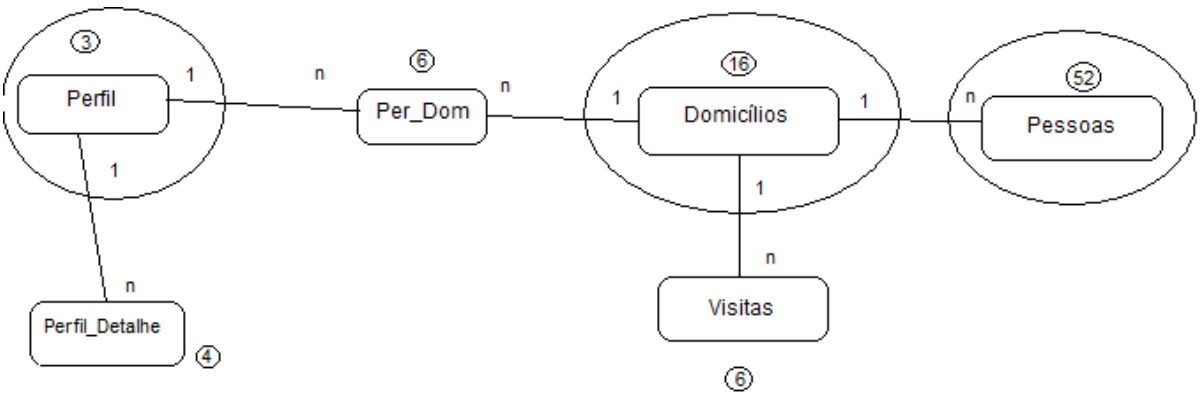


Figura 4 – DER da tarefa 1

Os números que estão circulados representam a quantidade de campos de cada tabela e os círculos maiores marcam as tabelas onde foi alterado o código de programação, logo, na hora de contar os pontos por função, contou-se somente dessas 3 tabelas. Esta tarefa deu um total de *12* pontos por função.

A tarefa 2, como não foi alterada a base de dados, e sim, alterada apenas a interface do sistema, teve como consequência uma quantidade de pontos por função menor. Como a *tarefa 2* só foi alterada a interface, calcula-se apenas o fator de ajuste, que são os níveis de influência sobre o tamanho total do programa. Ele pode aumentar ou diminuir em até 35% o número encontrado. Para se calcularem os níveis de influência, usa-se uma tabelinha (vide figura 3). Nessa tabela existem 14 itens e o que foi usado para calcular a *tarefa 2* foi o último item que é *facilidade de mudança* (vide figura 5). Após realizado o cálculo, a *tarefa 2* deu um total de 5 pontos por função.

14 - Facilidade de mudança	Considerando as seguintes características da aplicação, somar os pontos previsíveis, conforme haja identificação. Não havendo, a totalização será 0 (zero): • Flexibilidade de consulta a relatório permite tratar pedidos simples como, por exemplo, operadores e/ou aplicativos a apenas um arquivo lógico interno – **1 ponto**; • Flexibilidade de consulta a relatório permite tratar pedidos de complexidade média como, por exemplo, operadores e/ou aplicativos a mais de um arquivo lógico interno - **2 pontos**; • Flexibilidade de consulta a relatório permite tratar pedidos complexos como, por exemplo, combinações de operadores e/ou aplicativos a diversos arquivos lógicos internos - **3 pontos**; • Dados de controle do negócio são guardados em tabelas mantidas pelo usuário, com processos interativos *on-line*, mas mudanças só têm efeito no dia útil seguinte - **1 ponto**; • Dados de controle do negócio são guardados em tabelas mantidas pelo usuário, com processos interativos *on-line*, mas mudanças têm efeito imediato – **2 pontos**;

Figura 5 – Descrição do item facilidade de mudança

Agora que se tem o número de pontos por função de cada tarefa, é possível fazer uma comparação.

Em relação ao tempo total das tarefas, *a tarefa1*, que tem 12 pontos por função, levou 45 minutos para ser finalizada, já a *tarefa 2,* que tem 5 pontos por função levou-se menos tempo, 35 minutos, porém esta foi realizada em par. A *tarefa 1* possui 58,4% pontos por função a mais que a *tarefa 2*, se fosse para colocarmos o par para trabalhar na *tarefa 1*, a tarefa seria finalizada em 1h e 24 min, logo, o par trabalhando nessa tarefa, levaria mais tempo do que o *programador 1* levou para desenvolvê-la sozinho.

Em síntese, nesse caso, a programação em par não foi efetiva?

Para responder isso, foi analisado o pré-questionário respondido pelos dois programadores. Notou-se que o *programador 1* tem conhecimento de mais linguagens de programação do que o *programador 2*, incluindo a linguagem Visual Basic, já o *programador 2* conhece apenas a linguagem C/C++. No questionário, mostra também que o *programador 1* já trabalha com a linguagem Visual Basic em seus projetos, logo ele tem o conhecimento da arquitetura da linguagem, já o *programador 2* desenvolve sites e não tem contato com o Visual Basic.

Acompanhando o desenvolvimento da tarefa do par, percebeu-se que o *programador 1* teve o cuidado de explicar passo a passo para o *programador 2* o que deveria ser feito na *tarefa 2,* feito isso, o *programador 1* começou a explicar sobre o Visual Basic, a arquitetura da linguagem, a função de cada ferramenta, botões, o código fonte, isso fez com que demorasse mais para realizar a tarefa, porém o *programador 1* passou a idéia básica de funcionamento do programa para o *programador 2*, isso permitiu que os dois programadores pudessem trocar idéias em relação ao que deveria ser feito. Durante o desenvolvimento da tarefa, o *programador 2* passou a entender melhor do programa e começou a sugerir algumas idéias, inclusive a solução da *tarefa 2* partiu de uma idéia que o *programador 2* sugeriu, esta idéia foi trabalhada entre eles e chegou-se à solução. Em relação à inspeção de código, percebe-se que a programação em par funcionou, pois sempre que o *programador 1* digitava uma palavra errada no código, o *programador 2* sempre o avisava. Em relação à interrupção do trabalho para conversas em chats, checar emails, não ocorreu, porém isso não deu para ser avaliado corretamente, pois o tempo que levou para desenvolver a tarefa foi muito curto.

Logo abaixo, seguem os quadros com os pós-questionários respondidos pelos dois programadores após terem participado do estudo de caso:

Programador 1:

1) O que achou dos conceitos em geral da programação em par?

( ) Muito Ruim ( ) Ruim ( ) Regular (x) Bom ( ) Muito Bom

2) Você se sentiu à vontade trabalhando em par?

(x) Sim

( ) Não

3) Em relação a sua produtividade em par?

( ) Piorou

( ) Continuou mesma coisa

(x) Melhorou

4) Houve conflito de idéias entre você e seu par?

(x) Sim

( ) Não

5) Caso tenha tido um conflito de idéias, chegou-se a um consenso de qual seria a melhor opção a seguir?

(x) Sim

( ) Não

6) Em relação à inspeção de código, localização de erros e bugs no programa:

( ) Demorou-se mais para encontrar o erro

( ) O par não fez diferença por ter encontrado o erro

(x) Encontrou-se mais rápido o erro

7) Em relação à interrupção do trabalho para conversas em chats e checar e-mails, você ficou inibido(a) pelo seu par?

( ) Sim

(x) Não

8) Por estar programando em par, você aprendeu com seu companheiro e também ele conseguiu aprender com você?

(x) Sim

( ) Não

9) Qual foi o tempo total para a realização da tarefa?

R: Tarefa 1 levou 45 minutos e tarefa 2 levou 35 minutos.

10) Muitos profissionais da área não concordam com a programação em par, alegando que ao se usarem duas pessoas para fazer o trabalho de uma, logo se gastaria o dobro para a remuneração dos profissionais. Você concorda com essa alegação ou tem um pensamento diferente? Justifique sua resposta.

R: Na minha opinião, duas cabeças pensam melhor que uma. Criatividade é uma característica fundamental na programação, sendo assim, além de uma revisão de código em busca de erros, trabalhar em par pode facilitar a busca pela solução ideal de forma mais rápida.

Figura 6 – Quadro com o pós-questionário do *programador 1*.

Programador 2:

1) O que achou dos conceitos em geral da programação em par?

( ) Muito Ruim ( ) Ruim ( ) Regular (x) Bom ( ) Muito Bom

2) Você se sentiu à vontade trabalhando em par?

(x) Sim

( ) Não

3) Em relação a sua produtividade em par?

( ) Piorou

( ) Continuou mesma coisa

(x) Melhorou

4) Houve conflito de idéias entre você e seu par?

(x) Sim

( ) Não

5) Caso tenha tido um conflito de idéias, chegou-se a um consenso de qual seria a melhor opção a seguir?

(x) Sim

( ) Não

6) Em relação àinspeção de código, localização de erros e bugs no programa:

( ) Demorou-se mais para encontrar o erro

( ) O par não fez diferença por ter encontrado o erro

(x) Encontrou-se mais rápido o erro

7) Em relação à interrupção do trabalho para conversas em chats e checar e-mails, você ficou inibido(a) pelo seu par?

( ) Sim

(x) Não

8) Por você estar programando em par, você aprendeu com seu companheiro e também ele conseguiu aprender com você?

(x) Sim

( ) Não

9) Qual foi o tempo total para a realização da tarefa?

R: 45 minutos

10) Muitos profissionais da área não concordam com a programação em par, alegando que se ao usar duas pessoas para fazer o trabalho de uma, logo se gastaria o dobro para a remuneração dos profissionais. Você concorda com essa alegação ou tem um pensamento diferente? Justifique sua resposta.

R: Acho que pode ser uma forma de achar erros no momento da implementação e tornar a solução do problema mais rápida, não tenho muita certeza se sou a favor ou contra, porque não tinha um projeto grande para tirar muitas conclusões.

Figura 7 - Quadro com o pós-questionário do *programador 2*.

Não necessariamente programando em par, consegue-se desenvolver a tarefa mais rapidamente e foi isso o que ocorreu no estudo de caso, mas essa não é a única vantagem de se programar em par. Apesar de ser um simples estudo de caso, a programação em par mostrou-se eficiente para disseminar o conhecimento, troca de ídeias e a inspeção de código. Analisando os pós-questionários respondidos pelos programadores, nota-se que eles se sentiram à vontade programando em par e concordam que trabalhando dessa forma, consegue-se chegar à conclusão de um determinado problema mais rapidamente.

Para finalizar, na própria faculdade onde foi realizado esse estudo de caso e também na faculdade de Tecnologia de Ourinhos, o professor e doutor José Augusto Fabri realizou uma experiência sobre a programação em par adaptada do trabalho por Laurie Williams (http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF) e publicou em seu blog (www.engenhariasoftware.wordpress.com). Essa experiência nos ajuda a ter um melhor entendimento sobre os benefícios e desvantagens da programação em par.

Abaixo, segue a tabela da experiência realizada:

Todos os alunos desenvolveram um algoritmo de complexidade média, segundo o meu ponto de vista. O algoritmo tinha e classificar os números da diagonal secundária da uma matriz quadrada.																							
Medidas capturadas durante a experiência. M1 - Quantidade média de erros detectados durante o desenvolvimento do algoritmo. M2 - Quantidade média de testes de mesa efetuados. M3 - Tempo médio de desenvolvimento do algoritmo (unidade de medida – minutos) M4 - Percentual médio de programa que estavam rodando. Medida efetuada após a correção dos programas. Correção efetuada pelo professor da disciplina. Nota: Todas as medidas foram capturadas pelos alunos.																							
Instituições Participantes																							
Fundação Educacional do Município de Assis (FEMA)																Faculdade de Tecnologia de Ourinhos (FATEC)							
2007: 5º ano BCC – Eng. Sw II								2008: 5º ano BCC - Eng. SW II								2008: 6º semestre ASTI - Eng. Sw II							
Quantidade de alunos participantes e resultados inferidos																							
Total de alunos				21				Total de alunos				15				Total de alunos				33			
Total de pares				Total individual				Total de pares				Total individual				Total de pares				Total individual			
6				9				5				5				15				13			
M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
2.3	2.1	21	100	1.3	4.6	25	55	8.6	5.6	40	60	8.8	3.4	44	20	7	3	36	64	15	3	36	45

Tabela 1 – Informações mapeadas nas experiências. Em azul: Algoritmo desenvolvido e medidas capturadas. Em vermelho: Instituições participantes. **Em preto**: Total de alunos que desenvolveram o algoritmo, individualmente. Total dos alunos que trabalharam em par. **Fundo preto e letra branca**: Valor quantitativo mapeado na experiência.

Figura 8 – Tabela com a experiência realizada em salas de aula. (Tabela retirada do blog de José Augusto Fabri, www.engenhariasoftware.wordpress.com)

Segundo Fabri (www.engenhariasoftware.wordpress.com)

> Ao analisar os resultados, apresentados na Tabela , é possível constatar que os algoritmos desenvolvidos em par apresentam uma maior qualidade. Na experiência de 2007, 100% dos algoritmos desenvolvidos por dois programadores rodaram, contra 55% desenvolvidos individualmente. Em 2008, a tendência é a mesma. Um resultado que chama a atenção na tabela é o tempo de desenvolvimento, os pares foram mais rápidos que os programadores individuais, nas experiências desenvolvidas na FEMA. Na FATEC o tempo de desenvolvimento foi de 36 minutos para ambos. Na experiência desenvolvida por Constantine (1995) é possível verificar que o tempo de desenvolvimento dos pares tendem a cair com o passar do tempo, fato este confirmado junto aos alunos da FEMA.

5. CONCLUSÕES

5.1. RESULTADOS ALCANÇADOS

Com este trabalho de conclusão de curso, foi possível passar vários conceitos sobre a metodologia de desenvolvimento de software Extreme Programming e é claro, focando a prática da programação em par. Com o estudo de caso realizado, foi possível demonstrar os benefícios e desvantagens dessa prática e ela se mostrou mais positiva do que negativa. Práticas como a troca de idéias, inspeção do código, disseminação de conhecimento se mostraram eficientes durante o estudo de caso. Lembrando que esse trabalho não prova que a programação em par é a melhor solução para se trabalhar em uma empresa e sim dá indícios de que ela pode vir a funcionar em um ambiente de trabalho, porém isso vai depender muito do grupo de pessoas que a empresa tiver, pessoas que não se relacionam em grupo, que são individualistas, dificilmente conseguirão trabalhar com a programação em par.

Para empresas que tenham receio de usar tal prática, creio que valha a pena fazer um teste, pois essa pode ser uma forma de aumentar a capacidade produtiva da empresa por todos os benefícios que ela fornece.

5.2 TRABALHOS FUTUROS

Seria interessante, futuramente, realizar um experimento com a programação em par em um ambiente real de trabalho, com profissionais que já atuam na área há algum tempo e analisar os seus comportamentos. Para isso seria necessário que eles trabalhassem em um projeto relativamente grande, por cerca de um mês, aplicando todas as práticas da programação em par, para que se consiga tirar conclusões precisas, e com isso, ajudar outras empresas a aderirem à prática.

6. Referências Bibliográficas

ASTELS, David. **Test-driven development: a practical guide.** 1. ed. Upper Saddle River, NJ: Prentice Hall PTR, 2003.

BECK, Kent. **Extreme Programming explained: embrace change**. 1. ed. Reading, MA: Addison-Wesley, 2000.

BECK, Kent. **Test-driven development: by example.** 1. ed. Boston: Addison-Wesley, 2003.

BECK, Kent; ANDRES, Cynthia. Extreme **Programming explained: embrace change.** 2. ed. Upper Saddle River: Addison-Wesley, 2005.

BECK, Kent; FOWLER, Martin. **Planning Extreme Programming.** 1. ed. Boston: Addison-Wesley, 2001.

EMAM, Khaled E. **Finding success in small software projects.** Cutter Consortium Agile Project Management. Executive Report, v. 4, n. 11, 2003.

FABRI, José Augusto. **A Programação em Par (Pair Programming) pode melhorar a capacidade produtiva de uma empresa de desenvolvimento de software?.** www.engenhariasoftware.wordpress.com. Acesso em 06/11/2009.

FOWLER, Martin. **Refactoring: improving the design of existing code.** Upper Saddle River, NJ: Addison-Wesley, 2000.

JEFFRIES, Ron; ANDERSON, Ann; HENDRICKSON, Chet. **Extreme Programming installed. 1.** ed. Boston: Addison-Wesley, 2001.

L. L. Constantine, Constantine on **Peopleware. Englewood Cliffs**, NJ: Yourdon Press, 1995.

POPPENDIECK, Mary; POPPENDIECK, Tom. **Lean software development: an agile toolkit. 1.** ed. Upper Saddle River, NJ: Addison-Wesley, 2003

SOMMERVILLE, Ian; **Engenharia de Software.** 6.Ed. Tradução de Maurício de Andrade. São Paulo: Editora Pearson Education do Brasil, 2003.

TELES, Vinícius Manhães. **Extreme Programming.**http://improveit.com.br/xp/. Acesso em 20/06/2009 .

TELES, Vinícius Manhães. **Extreme Programming: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade.** 1. ed. São Paulo: Novatec, 2004.

TELES, Vinícius Manhães; **Um estudo de caso da adoção das práticas e valores do Extreme Programming**, dissertação de mestrado, 2005.

WEINBERG, Gerald M. **The psychology of computer programming.** 1. ed. New York: Van Nostrand Heinhold Company, 1971.

WILLIAMS, Laurie; KESSLER, Robert. **Pair programming illuminated.** 1. ed. Boston, MA: Addison-Wesley, 2003.

WILLIAMS, Laurie; KESSLER, Robert; CUNNINGHAM, Ward, et al. **Strengthening the case for pair programming**. IEEE Software, v. 17, n. 4, 2000.