UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Reginaldo Siqueira Júnior
Rodolfo Bueno de Oliveira

UM SIMULADOR PARA A EXECUÇÃO DE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Alfenas, 03 de fevereiro de 2014.

UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

UM SIMULADOR PARA A EXECUÇÃO DE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Reginaldo Siqueira Júnior
Rodolfo Bueno de Oliveira

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Rodrigo Martins Pagliares.

Alfenas, 03 de fevereiro de 2014.

Reginaldo Siqueira Júnior
Rodolfo Bueno de Oliveira

UM SIMULADOR PARA A EXECUÇÃO DE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Prof. Douglas Donizeti de Castilho Braz

Universidade Federal de Alfenas

Prof. Eduardo Gomes Salgado

Universidade Federal de Alfenas

Prof. Rodrigo Martins Pagliares (Orientador)

Universidade Federal de Alfenas

Alfenas, 03 de fevereiro de 2014.

Dedicamos este trabalho à Deus e a todos nossos familiares

AGRADECIMENTO

Agradeço ao Prof. Orientador Rodrigo Martins Pagliares, por ter me aceito como orientando e por nos guiar no desenvolvimento deste trabalho.

Ao Reginaldo Siqueira Junior por ter desenvolvido esta monografia em parceria comigo e finalmente agradeço acima de tudo a minha família, que me apoia, me ama e que sempre vivera junto a mim para todo o sempre.

Rodolfo

Agradeço primeiramente a Deus, por ter me abençoado e iluminado meu caminho durante todo esse trajeto na Universidade. Por ter me dado força nos momentos complicados e por ter me dado sabedoria nos momentos de escolhas difíceis. E principalmente por ter cuidado bem da minha família, nos momentos em que eu estava ausente.

Aos meus pais, Reginaldo e Vani, e minha irmã Ana Laura, pelas orações e apoio incondicional nos momentos de dificuldade, onde me apoiaram e me fizeram seguir em frente. Sem o apoio deles, seria impossível alcançar esse objetivo. Essa conquista, com certeza, é nossa.

Aos meus amigos de sala e companheiros de república, que estiveram comigo durante todo esse trajeto.

Aos professores do curso, pelo conhecimento que me passaram durante todos esses anos. Em especial aos também professores, mas principalmente amigos, Douglas e Humberto, pela oportunidade de trabalhar juntamente com eles no Laboratório de Pesquisa e Desenvolvimento (LP&D). Esta oportunidade foi essencial no meu crescimento como profissional.

Ao professor e orientar Rodrigo Martins Pagliares, por ter nos dado suporte e apoio durante todo o desenvolvimento deste trabalho.

Ao Rodolfo Bueno de Oliveira por ter me auxiliado no desenvolvimento desta Monografia.

Reginaldo

"Maior que a tristeza de não ter vencido é a vergonha de não ter lutado."

Rui Barbosa

RESUMO

As organizações de software têm como principal objetivo produzir software de alta qualidade, sempre procurando uma boa relação entre custo benefício e prazo de entrega. Para que este objetivo seja concluído é preciso escolher um dos Modelos de Processos de Desenvolvimento de Software que descrevem a "vida" do software desde a concepção, implementação, entrega, utilização e manutenção. O uso de simulação para estimar qualitativamente e verificar a eficácia de um determinado processo de desenvolvimento de software muitas vezes representa uma solução alternativa válida. Um simulador de processos de desenvolvimento de software é proposto por este trabalho, capaz de auxiliar as empresas no desenvolvimento de software indicando qual prática de desenvolvimento utilizar em conjunto com o scrum.

Palavras-Chave: Software, Modelos de Processo de Desenvolvimento de Software, Simulação, Scrum.

ABSTRACT

Software organizations have as their main objective to produce high quality software, always looking for a good value for money and delivery time. For this goal to be completed you must choose one of the Models of Software Development Processes that describe the "life" of software from design, implementation, delivery, use and maintenance. The use of simulation to estimate qualitatively and verify the effectiveness of a given process of software development is often a valid workaround. A process simulator software development is proposed in this work, able to assist companies on software development indicating which development practice used in conjunction with scrum.

Keywords: Software Process Models of Software Development, Simulation, Scrum.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

DDT	Desenvolvimento Dirigido por Testes
MPDS	Modelo de Processo de Desenvolvimento de Software
SPS	Simulador de Processos de Software
XP	Extreme Programming
L	Linhas de Código
C	Classes
M	Métodos

SUMÁRIO

1
Introdução

Este capítulo apresenta as razões pelas quais este trabalho foi desenvolvido, bem como o que este pretende resolver e a proposta de como a resolução será feita. Na seção 1.1 são apresentadas as justificativas para o desenvolvimento deste trabalho. A seção 1.2 apresenta o problema central que este trabalho procura resolver. Na seção 1.3 são apresentados os objetivos que se pretende atingir com este trabalho. E a seção 1.4 apresenta a Organização da Monografia.

O scrum é uma metodologia de desenvolvimento de software muito utilizada nos últimos anos. No entanto, ele não descreve qual prática de desenvolvimento utilizar (programação em pares, desenvolvimento dirigido por teste, entre outras). Um software pode ser definido como um conjunto de procedimentos, dados e documentação, associados a um sistema de computador, que não se desgasta. A maior parte dos softwares são feitos sob medida (PRESSMAN, 1995).

A engenharia de software é definida então como a aplicação dos princípios da engenharia em todos os aspectos da produção de software (PAULA, 2009), (SOMMERVILLE, 2007). Três elementos fundamentais que compõe a engenharia de software são: métodos, que são as atividades a serem realizadas; ferramentas que oferecem suporte a realização de uma ou mais atividades definidas nos métodos; e procedimentos, que são uma combinação de métodos e ferramentas responsáveis por definir a sequencia de aplicação de métodos.

Segundo (PFLEEGER, 2004) um paradigma representa uma abordagem ou filosofia em particular para construção de um software. Através da utilização de um paradigma, e a escolha de um modelo de processo de desenvolvimento de software - MPDS, (PAULA, 2009) descreve os modelos de ciclo de vida divididos da seguinte forma: Ciclos de vida em cascata; Ciclos de vida em espiral entre outros.

O principal objetivo desta monografia é avaliar como as práticas-chave influenciam a evolução de um determinado projeto. Um simulador de processos de software – SPS foi desenvolvido na linguagem de programação java, capaz de variar o nível de utilização das práticas de desenvolvimento.

1.1 Justificativa e Motivação

A simulação de processos de software é muito utilizada atualmente, como visto no trabalho Software Process Simulation Model XP (MELIS, 2006), onde foi desenvolvido um SPS que avalia as práticas de desenvolvimento envolvidas no MPDS Extreme Programming (XP). Assim, resolveu-se pela construção de um simulador que possa contribuir para as empresas de software de diferentes formas: financeiramente, na eficácia do processo, no tempo de execução, entre outros.

A principal motivação deste trabalho foi desenvolver um software capaz de informar resultados pertinentes em relação às práticas de desenvolvimento de software, para que possam ser realizadas futuras comparações.

1.2 Problematização

Uma situação muito comum de se observar dentro das organizações de software é a alteração do processo inicialmente definido, isto ocorre devido ao fato do surgimento de novos eventos durante a execução do projeto. O surgimento de um determinado evento pode ocasionar uma modificação no processo corrente. Uma das causas podem ser as diferentes características do projeto, como por exemplo: a quantidade de desenvolvedores da equipe; as praticas utilizadas; o tempo do projeto em dias; entre outros.

Neste contexto deseja-se saber:

- Como construir um simulador que possibilite a simulação da execução de um processo de desenvolvimento de software iterativo e incremental para gerenciamento de projetos e desenvolvimento ágil de software?

1.3 Objetivos

1.3.1 Gerais

Este trabalho tem como objetivo geral avaliar qual o conjunto de práticas de desenvolvimento melhor se adéqua ao framework para gestão de projetos scrum. Tal avaliação torna-se necessária tendo em vista que o scrum não determina as práticas de desenvolvimento a serem utilizadas em seu ciclo de vida.

1.3.2 Específicos

Para que o objetivo geral deste projeto seja atingido é necessário desenvolver um simulador para que seja possível ao final de sua execução, analisar as técnicas de desenvolvimento programação em pares e desenvolvimento dirigido por teste - DDT melhor se adequa ao framework de desenvolvimento de software scrum.

1.4 Organização da Monografia

Este trabalho encontra-se organizado da seguinte forma: No Capítulo 2 apresenta-se todo o referencial teórico necessário para o entendimento desta monografia; No Capítulo 3 são apresentados os trabalhos relacionados a esta monografia; No Capítulo 4 é apresentado o Simulador de Desenvolvimento de Software; No Capítulo 5 são apresentados os resultados obtidos a partir do processo de simulação; No Capítulo 6 é apresentada a conclusão da monografia, além de todos os detalhes dos trabalhos futuros que podem vir a ser realizados a partir deste.

2
Revisão Bibliográfica

Este capítulo apresenta o estado da arte dos conceitos necessários para um melhor entendimento desta monografia. Inicialmente é definido alguns termos de Engenharia de Software. Posteriormente apresenta-se o scrum, programação em pares, DDT, MPDS, respectivamente.

2.1 Scrum

Desenvolvido por Ken Schwaber e Jeff Sutherland, o scrum é um framework ou uma estrutura processual que suporta o desenvolvimento e manutenção de produtos complexos, permitindo as pessoas: tratar, desenvolver, manter e resolver diversos tipos de problemas, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível (SCHWABER e SUTTHERLAND, 2011).

Fundamentado nas teorias empíricas de controle de processo, ou empirismo (o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido), o scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. Três pilares apoiam a implementação de controle de processo empírico: transparência, inspeção e adaptação (SCHWABER e SUTTHERLAND, 2011).

O framework é composto por diversos componentes, cada um com um propósito específico. Estes componentes são essenciais para o uso e sucesso do scrum. O funcionamento básico do scrum pode ser visto na Figura 1, onde temos o backlog da sprint que são as funcionalidades atribuídas para sprint através do backlog do produto. O processo inicia-se com uma reunião de planejamento da sprint onde os itens são divididos por equipes. Ao final da sprint, após 30 dias, ocorre a reunião de revisão da sprint e a entrega de novas funcionalidades.

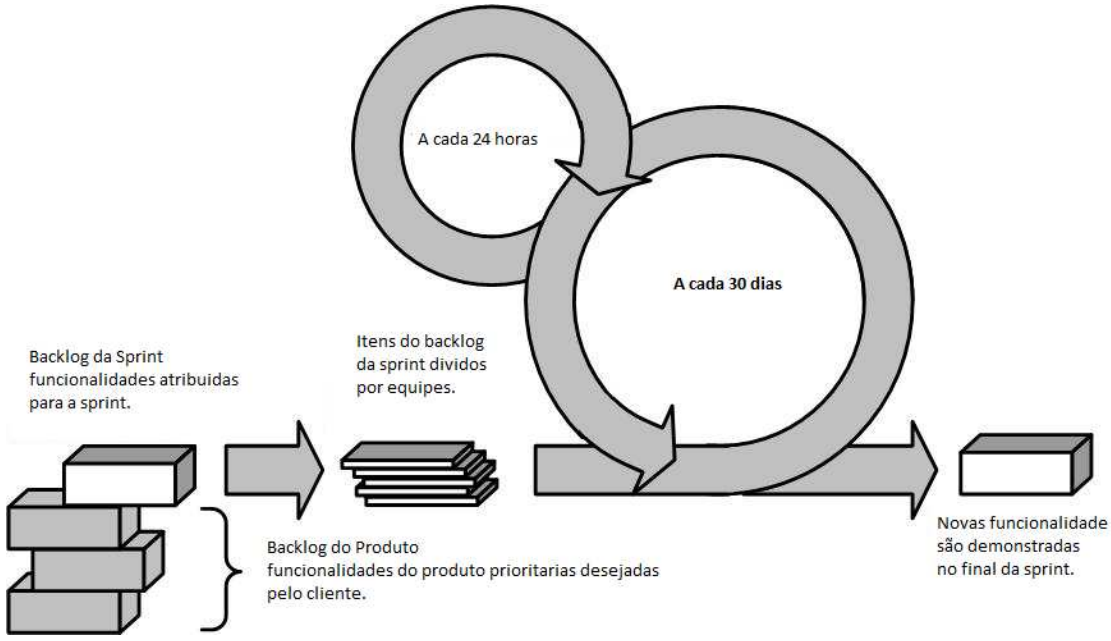

Figura 1 - Funcionamento do Scrum

2.1.1 Time Scrum

O scrum é composto pelo product owner, a equipe de desenvolvimento e o scrum master. Os times scrum são auto organizáveis (escolhem qual a melhor forma para completarem seu trabalho sem a interferência de outros fora da equipe) e multifuncionais (possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe).

O time scrum tem como principal objetivo entregar produtos de forma iterativa e incremental, maximizando as oportunidades de realimentação. As entregas incrementais de um produto garantem que uma versão potencialmente funcional do produto esteja sempre disponível (SCHWABER e SUTTHERLAND, 2011).

2.1.1.1 Product Owner

O product owner, conhecido como o dono do negócio, é o responsável por maximizar o valor do produto, o trabalho da equipe de desenvolvimento e por gerenciar o backlog do produto, que inclui: expressar claramente os itens pertencentes a ele e ordená-los de forma a alcançar melhor as metas e missões,

garantindo que o backlog do produto seja de fácil entendimento a todos e também garante o trabalho realizado pelo time de desenvolvimento.

Ele pode fazer o trabalho acima, ou delegar para a equipe de desenvolvimento fazê-lo. No entanto, continua sendo o responsável pelos trabalhos. Alterações nas prioridades dos itens escolhidos devem convencer o product owner. Assim para que ele obtenha sucesso dentro da organização em que trabalha, toda a organização deve respeitar as suas decisões (SCHWABER e SUTTHERLAND, 2011).

2.1.1.2 Equipe de Desenvolvimento

O modelo de equipe no scrum foi projetado para aperfeiçoar a flexibilidade, criatividade e a produtividade. Consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto ao final de cada sprint, ou seja, o tempo no qual a equipe de desenvolvimento tem para se produzir uma nova versão do produto.

As equipes de desenvolvimento são estruturadas, auto organizáveis, multifuncionais e autorizadas pela organização a gerenciar seu próprio trabalho. Individualmente os integrantes da equipe de desenvolvimento podem ter diferentes habilidades e áreas de especialização, mas a responsabilidade pertence à equipe de desenvolvimento como um todo.

O tamanho ideal da equipe de desenvolvimento deve ser pequeno o suficiente para manter a equipe ágil, e grande o suficiente para se completar uma parcela significativa do trabalho. Entre três e nove, é uma margem adequada para o tamanho da equipe de desenvolvimento. Os papéis de product owner e de scrum master não são incluídos nesta contagem, a menos que, eles também executem o trabalho do backlog da sprint (SCHWABER e SUTTHERLAND, 2011).

2.1.1.3 Scrum Master

O entendimento e aplicação do scrum são de responsabilidade do scrum máster. Ele faz isso para garantir que o time scrum possa aderir à teoria, pratica e regras do scrum. Ele pode ser considerado como um servo-líder, pois ele facilita o

entendimento das interações para aqueles que não fazem parte do time scrum e estão interessados no negócio.

O scrum master trabalha para o product owner realizando diversas tarefas como por exemplo: ensinando o time scrum a criar, compreender e manipular de forma ágil o backlog do produto, encontrando técnicas para o gerenciamento efetivo e por fim facilitando os eventos do scrum conforme exigidos ou necessários.

O scrum master trabalha para a equipe de desenvolvimento realizando diversas tarefas como por exemplo: treinar, ensinar e liderar a equipe de desenvolvimento em autogerenciamento, interdisciplinaridade e na criação de produtos de alto valor, removendo impedimentos que alterariam o progresso da atual da equipe de desenvolvimento.

O scrum master trabalha para a organização realizando diversas tarefas, por exemplo: ele lidera, treina e planeja a implementação do scrum dentro da organização trabalhando com outro scrum master ou de forma individual. Também ajuda os funcionários e partes interessadas a compreender e tornar aplicável o scrum, aumentando sua eficácia (SCHWABER e SUTTHERLAND, 2011).

2.1.2 Artefatos

Artefato é um produto que representa o trabalho realizado pela equipe de desenvolvimento, os artefatos definidos no scrum são especificamente projetados para maximizar a transparência das informações necessárias para assegurar que o time scrum obtenha sucesso na entrega do incremento do produto.

2.1.2.1 Backlog do Produto

Este artefato é usado para descrever o trabalho previsto para o produto, é uma lista ordenada de todos os requisitos necessários no produto, mas nunca está completo. Os desenvolvedores apenas estabelecem os requisitos inicialmente conhecidos e melhor entendidos, assim o backlog do produto evolui tanto quanto o produto, existindo em quanto o produto existir. É dinâmico, mudando constantemente para identificar o que o produto necessita para ser mais apropriado, competitivo e útil.

Os itens no topo da lista ordenada do backlog do produto determinam as atividades de desenvolvimento mais imediatas e são mais claros e detalhados que os itens de ordem mais baixa. Quanto maior a ordem (topo da lista) de um item, mais o item será considerado.

Os itens do backlog do produto que irão ocupar o desenvolvimento na próxima sprint são mais refinados, tendo sido decompostos de modo que todos os itens possam ser completados ao final da sprint. Estes itens podem ser considerados como "disponíveis" ou "executáveis" para seleção na reunião de planejamento da sprint.

Assim um produto que é utilizável, ganha valor e o mercado oferece retorno. O backlog do produto torna-se uma lista maior e mais completa, e muitas vezes, é por esta razão que este artefato é considerado como um artefato emergente (SCHWABER e SUTTHERLAND, 2011).

2.1.2.2 Backlog da Sprint

O conjunto de itens do backlog do produto selecionados para a sprint é denominado backlog da sprint. É uma previsão da equipe de desenvolvimento sobre qual funcionalidades poderão ser realizar dentro dos limites de tempo definidos pela organização no próximo incremento e o trabalho necessário para entregar tais funcionalidades.

Este artefato define qual o trabalho a equipe de desenvolvimento realizará para converter os itens do backlog do produto em um incremento utilizável do produto. Em casos onde é preciso realizar alguma modificação do backlog da sprint durante a execução de uma sprint, estas devem ser realizadas pela equipe de desenvolvimento. Porém estas modificações raramente são necessárias.

2.1.3 Eventos

No scrum os eventos são do tipo *time-box*, onde todo evento tem uma duração máxima e são usados para criar uma rotina e minimizar a necessidade de reuniões não definidas. Isto garante que uma quantidade adequada de tempo seja gasta sem ocasionar perdas desnecessárias de tempo no processo de planejamento.

Cada evento no scrum é uma oportunidade de inspecionar e adaptar alguma coisa. Estes eventos são especificamente projetados para permitir uma transparência e inspeção criteriosa. A não inclusão de qualquer um dos eventos resultará na redução da transparência e da perda de oportunidade para inspecionar e adaptar (SCHWABER e SUTTHERLAND, 2011).

2.1.3.1 Sprint

Uma sprint pode ser considerada como um horizonte ou *time-box* de não mais que um mês, durante o qual uma nova versão do produto é criada. Cada sprint contém a definição do que é para ser construído, ou seja, um plano projetado e flexível que irá guiar a construção, o trabalho e o resultado do produto.

Durante a sprint não são feitas mudanças que podem afetar o seu objetivo principal. A composição da equipe de desenvolvimento permanece constante, as metas de qualidade não diminuem e o escopo pode ser refinado e renegociado entre o product owner e a equipe de desenvolvimento.

Uma sprint pode ser cancelada antes do seu término, mas somente o product owner tem a autoridade para cancelá-la, embora ele (ou ela) possa fazer isso sob influência das partes interessadas, da equipe de desenvolvimento ou do scrum master. A sprint poderá ser cancelada se o objetivo dela se tornar obsoleto. Isto pode ocorrer se a organização mudar sua direção ou se as condições do mercado ou das tecnologias mudarem. Geralmente a sprint deve ser cancelada se ela não faz mais sentido das questões do projeto. No entanto, devido a sua curta duração, raramente cancelamentos fazem sentido.

Quando ela é cancelada, qualquer item de backlog do produto completado é revisado. Se uma parte do trabalho estiver potencialmente utilizável, tipicamente o product owner o aceita. Todos os itens de backlog do produto incompletos são estimados novamente e colocados de volta na lista de requisitos. O trabalho feito se deprecia rapidamente e deve ser frequentemente refeito. O cancelamento de uma sprint consome muitos recursos, já que todos os membros do time scrum devem se reagrupar em outra reunião de planejamento da sprint para iniciar uma nova sprint.

2.1.3.2 Reunião de Planejamento da Sprint

O trabalho a ser realizado na sprint é decidido nesta reunião, onde uma meta é criada com o trabalho colaborativo de todo o time scrum. Esta reunião pode ser dividida em duas partes, na primeira parte a equipe de desenvolvimento trabalha para prever as funcionalidades que serão desenvolvidas durante a sprint. O product owner apresenta os itens de backlog do produto ordenados para a equipe de desenvolvimento e todo o time scrum colabora com o entendimento do trabalho da sprint. Somente a equipe de desenvolvimento pode avaliar o que pode ser completado ao longo da próxima sprint.

Durante a segunda parte da reunião de planejamento da sprint a equipe de desenvolvimento decide como irá construir essas funcionalidades durante a sprint e transformá-las em um incremento de produto. Os itens de backlog do produto selecionados, junto com o plano de entrega destes itens são chamados de backlog da sprint.

O product owner pode estar presente durante a segunda parte da reunião de planejamento da sprint. A presença dele pode ser útil para ajudar nas decisões conflituosas de troca de itens selecionados para a sprint atual. E em casos onde a equipe de desenvolvimento determina que exista excesso ou falta de trabalho, os itens do backlog da sprint podem ser renegociados com o product owner. Existe a possibilidade de outras pessoas participarem desta reunião de forma a fornecer opinião técnica ou de domínios específicos.

2.1.3.3 Reunião Diária

É uma reunião de 15 minutos, para que a equipe de desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última reunião, e prever o trabalho que deverá ser feito antes da próxima reunião diária. Sempre ocorrem no mesmo horário e local.

A equipe de desenvolvimento usa esta reunião para avaliar o progresso em direção ao objetivo da sprint e para avaliar se o progresso tende a completar o trabalho do backlog da sprint. A reunião diária aumenta a probabilidade da equipe de desenvolvimento atingir o objetivo da sprint.

O scrum master assegura que a equipe de desenvolvimento tenha a reunião, mas é a própria equipe de desenvolvimento que fica responsável por conduzi-la. O scrum master ensina a equipe de desenvolvimento a manter a reunião diária dentro do seu limite de tempo.

As reuniões diárias melhoram as comunicações, identificam e removem impedimentos para o desenvolvimento, promovem rápidas tomadas de decisão e melhoram o nível de conhecimento da equipe de desenvolvimento. Esta é uma reunião chave para inspeção e adaptação (SCHWABER e SUTTHERLAND, 2011).

2.1.3.4 Reunião de Revisão da Sprint

A revisão da sprint é executada no final da sprint para inspecionar o incremento e adaptar o backlog do produto se necessário, com duração de quatro horas. Durante a reunião, o time scrum e as partes interessadas discutem sobre o que foi feito na sprint. A apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração.

Durante a reunião de revisão da sprint o product owner identifica o que foi feito e projeta as prováveis datas de conclusão baseado no progresso até a data atual. A equipe de desenvolvimento discute sobre seus feitos durante a sprint, quais problemas ocorreram e como estes foram resolvidos.

O grupo todo colabora sobre o que fazer a seguir, e é assim que a reunião de revisão da sprint fornece valiosas entradas para a próxima reunião de planejamento da sprint. Obtém-se como resultado desta reunião um backlog do produto revisado que define o provável backlog do produto para a próxima sprint.

2.1.3.5 Reunião da Retrospectiva da Sprint

Com duração de três horas para uma sprint de um mês. A reunião da retrospectiva da sprint é uma oportunidade para o time scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima sprint. O propósito desta reunião é: verificar como a última sprint ocorreu em relação às pessoas, processos e ferramentas, identificar e ordenar os principais itens que foram implementados e as potenciais melhorias, e criar um plano para implementar melhorias de modo que o time scrum desenvolve seu trabalho.

Durante cada reunião de retrospectiva da sprint, o time scrum planeja formas de aumentar a qualidade do produto. Ao final da retrospectiva da sprint, o time scrum deverá ter identificado melhorias que serão implementadas na próxima sprint. A retrospectiva da sprint fornece um evento dedicado e focado na inspeção e adaptação. No entanto, as melhorias podem ser adotadas a qualquer momento (SCHWABER e SUTTHERLAND, 2011).

2.2 Técnicas de Desenvolvimento de Software

Existem diversas técnicas de desenvolvimento de software, bem como diversos MPDS. Quando se fala nesses assuntos é inevitável citar Kent Beck, que foi o criador do *XP* e do DDT.

2.2.1 Programação em Pares

Quando a codificação é realizada por um par de desenvolvedores, trabalhando em um mesmo computador e na mesma tarefa é denominado programação em pares. Os benefícios obtidos quando se utiliza esta técnica podem ser diversos, como por exemplo: a implementação do sistema se torna mais rápida, acelerando o desenvolvimento do produto e a remoção erros. Uma revisão contínua é realizada pelos desenvolvedores, onde erros são removidos o mais cedo possível no ciclo de desenvolvimento de software, ou seja, estes erros têm um impacto menor no cronograma e custos.

A utilização de programação em pares aumenta a aprendizagem e leva a uma melhor distribuição do conhecimento entre os membros da equipe, porque todo mundo trabalha com todo mundo durante o projeto. Logo se torna uma ótima oportunidade de introduzir novos membros da equipe no projeto.

Ao se utilizar programação em pares um desenvolvedor fica responsável pela a produção de código. O outro pode pensar mais estrategicamente sobre as implicações do projeto e pode considerar alternativas e soluções, como por exemplo o melhoramento do design.

2.2.2 Desenvolvimento Dirigido por Testes (DDT)

Quando os desenvolvedores escrevem o teste antes do próprio código, esta técnica é denominada desenvolvimento dirigido por testes. Através de um ciclo rápido onde os testes são adicionados e refatorados para limpar o código. Desta forma o projeto de software evolui através dos testes.

Segundo (CRISPEN e HOUSE, 2003) todos os testes devem ser automatizados. As melhorias na qualidade alcançada pela equipe de desenvolvimento variaram de 38% a 267, reduzindo os defeitos. Por outro lado, o processo de testes durante a codificação levou mais tempo (YNCHAUSTI, 2001).

(BOBY e LAURIE, 2003) realizaram um experimento com dois grupos: um utilizando DDT e o outro utilizou uma abordagem em cascata. O experimento mostrou que os desenvolvedores DDT levaram 16% mais de tempo do que aqueles que não usaram essa prática, mas produziram um código de maior qualidade.

(WILLIAMS *et al*, 2003) realizaram um estudo de caso para investigar os efeitos do DDT na IBM. Concluíram que a introdução dessa prática alcança uma redução de 40% de defeitos e a equipe apresentou mais de 0.5 linhas de código de teste para cada linha de código de implementação.

(ERDOGMUS, MORISIO e TORCHIANO, 2005) realizaram seus experimentos e observaram que a escrita de mais testes conduziu a um nível mais elevado de produtividade.

2.3 Modelagem de Simulação de Processo de Desenvolvimento de Software

Um modelo é uma representação simplificada, por exemplo, a reprodução de algo, que pode ser uma pessoa, coisa, procedimento, entre outros. A modelagem é feita a fim de se obter um exemplo que possa fornecer informações e características importantes do sistema que se deseja estudar, prever, modificar ou controlar.

Logo, um MPDS é um sistema computadorizado que possui as características acima descritas e que é a representação de algum sistema real. Uma das motivações para desenvolver um modelo de simulação é a obtenção de informações importantes quanto a custos, riscos, entre outras variáveis (RUS, HALLING e BIFFL, 2003).

Os usos mais comuns de MPDS são para fornecer uma base para a experimentação, prever o comportamento, planejamento, ajuda na tomada de decisões, na redução de riscos, na gestão nos níveis estratégicos, táticos e operacionais.

Em relação às organizações de processo de desenvolvimento de software, os ganhos podem ser diversos quando se utiliza um modelo de simulação. Um exemplo é que ajuda no aprendizado das pessoas envolvidas na organização. Estas pessoas podem ser: gerentes de projeto, desenvolvedores de software, entre outros.

O SPS está se tornando cada vez mais popular na comunidade de engenharia de software, tanto entre acadêmicos quanto profissionais (KELLNER, MADACHY, RAFFO, 1999). Na verdade, estão sendo constantemente desenvolvidas novas e inovadoras técnicas de engenharia de software para uma melhor compreensão dos processos.

A simulação é útil para avaliar a eficácia e prever possíveis problemas. A SPS também pode ajudar os gerentes de projetos e engenheiros de processo para planejar mudanças no processo de desenvolvimento. O desenvolvimento de um modelo de simulação é uma forma relativamente barata em comparação a projetos de software reais quanto aos custos, riscos e a complexidade do sistema real.

Dados reais referentes a qualquer processo de desenvolvimento de software podem ser usados para calibrar o modelo, e os resultados do processo de simulação são usados para o planejamento, projeto e análise de experimentos reais (RUS, HALLING, BIFFL, 2003).

2.3.1 Técnicas de Simulação

Durante a concepção e implementação de um simulador, podem ser utilizadas diversas técnicas e estratégias para modelar o comportamento de um determinado

sistema. Fatores como o nível de abstração e da precisão desejada e velocidade da simulação, devem ser levados em consideração ao se projetar o kernel do simulador (CRAIG, 1996).

O sistema que está sendo simulado pode ser dividido em duas categorias distintas, dependendo do grau de aleatoriedade associado com o comportamento do sistema no seu ambiente simulado. Um sistema que apresente comportamento aleatório é definido como um sistema estocástico. Por outro lado, um sistema de simulação determinístico é aquele que não incorpora um comportamento aleatório.

Na literatura são definidos vários modelos de simulação, como o modelo de simulação continua, onde o estado do sistema é representado por variáveis dependentes que mudam continuamente no tempo. Também são encontrados diversos outros modelos como: modelo de simulação mista, determinística e estocástica, modelo de analise de sensibilidade e também o modelo utilizado no desenvolvimento deste trabalho que é o modelo de simulação de eventos discretos.

2.3.2 Modelo de Simulação de Eventos Discretos

Segundo (FISHMAN, 2001) a simulação de eventos discretos envolve a modelagem de um sistema à medida que progride ao longo do tempo e é particularmente útil para a análise de sistemas de filas. Tais sistemas utilizam o conceito de "tempo simulado", o que caracteriza a simulação de eventos discretos.

Cada evento esta vinculado há um tempo simulado e esta na fila de eventos esperando a sua chamada para a execução. Enquanto existirem eventos na fila o simulador entra no seu laço principal executando as seguintes operações:

1. Retira o item mais prioritário da fila de eventos.
2. Atribui à variável tempo atual o valor do tempo simulados, associados ao evento atual.
3. Trata o evento atual.

Cada evento na lista de eventos tem uma referência para a lógica do modelo que tem de ser executada no momento correto. Isto implica que mais de um evento pode fazer referência a mesma lógica do modelo, significando que a mesma lógica

é usada muitas vezes durante todo o tempo de execução (FISHMAN, 2001), (KREUTZER, 1986), (BALL, 2001), (SCHRIBER, BRUNNER, 1999).

A simulação acaba quando a fila de eventos estiver vazia. Ao final da simulação, dados são impressos para que seja possível fazer comparações com os dados dos processos de desenvolvimento de software reais passados como parâmetros para o simulador. Para uma melhor compreensão, o fluxo de simulação de um simulador discreto pode ser visto da Figura 2.

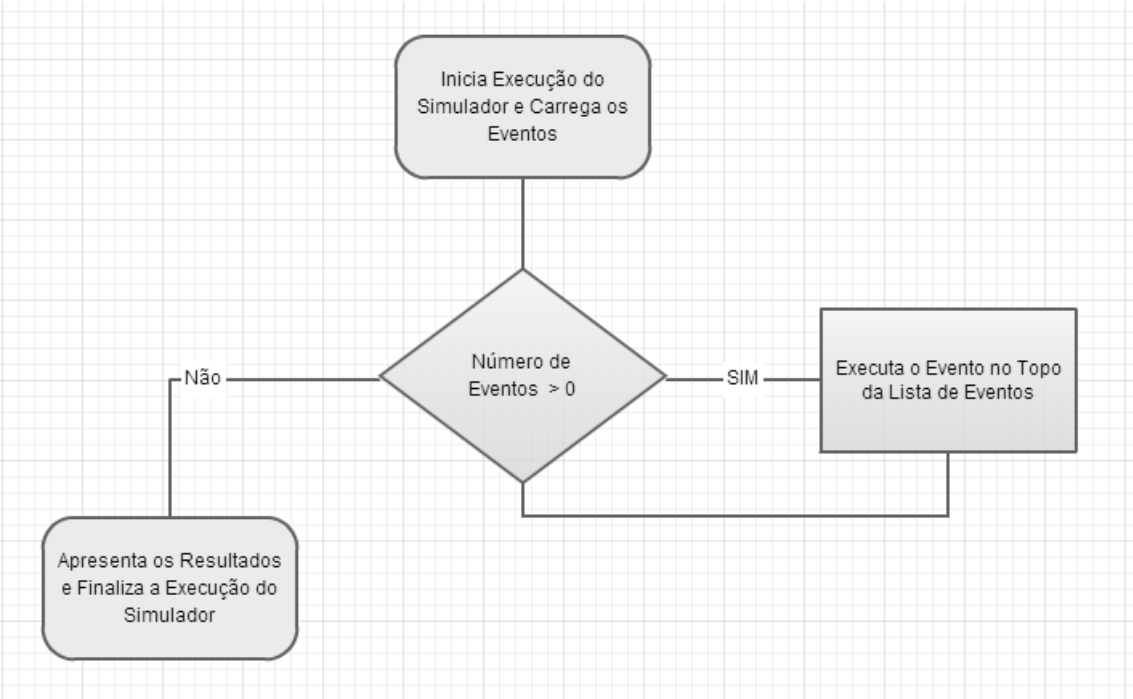

Figura 2 - Fluxo de um Simulador de Eventos Discretos

3
Trabalhos Relacionados

Alguns trabalhos relacionados ao proposto aqui foram divulgados na literatura como (ALISTAR e LAURIE, 2000), onde uma sessão de desenvolvimento realizada com programação em pares é mais eficiente do que uma "sessão de desenvolvimento com programação solo", em termos de tempo necessário para implementar uma única história de usuário e defeitos injetados. Assume-se que a velocidade da programação em pares com dois desenvolvedores é aumentada em 40%, como pode ser visto em alguns estudos (ALISTAR e LAURIE, 2000), (JOHN, 1998).

Um estudo completo e aprofundado pode ser visto em (BOBY e LAURIE, 2003), onde pode-se ver que desenvolvedores que utilizam DDT levam 16% a mais de tempo para desenvolver um determinado projeto, do que desenvolvedores que não utilizam DDT para desenvolver o mesmo projeto. Desse modo pode-se afirmar que a velocidade diminui em 14% quando uma sessão de DDT é utilizada no momento do desenvolvimento.

O simulador desenvolvido nesta monografia, têm como base fundamental a sessão de desenvolvimento descrita por (MELIS, 2006), onde são simulados todos os passos presentes na prática denominada XP, desde o planejamento até a sessão de desenvolvimento e as sessões de refatoração. No trabalho de (MELIS, 2006) algumas práticas de desenvolvimento também são avaliadas. Sendo assim, este trabalho pode ser considerado como a maior referência para o desenvolvimento desta monografia.

4
O Simulador de Processos de Desenvolvimento de Software

4.1 Características Especificas do Modelo de Simulação

Para que seja possível fazer uma avaliação sobre as melhores práticas a serem utilizadas no framework para gestão de projetos scrum, um simulador foi desenvolvido. Este simulador trabalha com as características do scrum e foi desenvolvido na linguagem de programação java. A escolha dessa linguagem foi feita por diversos motivos, dentre os principais sua simplicidade, facilidade de uso e grande quantidade de recursos para o desenvolvimento.

A técnica de simulação utilizada é a simulação de eventos discretos. Essa técnica permite uma grande facilidade na definição do modelo, fazendo com que a simulação seja muito semelhante ao que ocorre no dia a dia em projetos reais, onde as ações ocorrem em um espaço de tempo com eventos pré-definidos e ordenados.

Esta técnica de simulação é necessária e essencial, pois permite que as entidades sejam bem definidas e possuam suas próprias características básicas, estas que as diferem umas das outras. Sabe-se que as atividades realizadas pelas entidades durante a simulação fazem com que ocorram atualizações e mudanças no modelo, ou seja, algumas ações são determinantes durante a simulação. Sendo assim, percebe-se o maior realismo do simulador que facilita o entendimento de como o processo ocorre e qual a importância de cada entidade durante a execução do mesmo.

Um exemplo da grande realidade dessa simulação é que assim como em projetos reais, os desenvolvedores tem suas habilidades melhoradas com o passar do tempo em um projeto. Isso pode ser visto de forma natural no mundo real, pois quanto mais realizamos uma determinada tarefa, melhor será nosso desempenho em tal atividade. Ao final de cada sessão de desenvolvimento o time ganha em muitos aspectos, devido à evolução constante dos seus desenvolvedores.

4.2 Descrição do Modelo

Este modelo não simula as reuniões feitas no scrum, pois seu foco está no comparativo das melhores práticas para serem utilizadas neste framework. Sendo assim, através de parâmetros iniciais as reuniões e definições feitas no inicio do projeto funcionam como se já tivessem sido realizadas, deixando assim, o foco para o desenvolvimento.

O objetivo da simulação é fazer com que a equipe desenvolva no dia a dia todas as funcionalidades definidas no backlog do produto, tendo um prazo determinado por uma quantidade limitada de dias do projeto e horas de sessão de desenvolvimento por dia. As atividades de desenvolvimento são realizadas pelos atores. O modelo possui as seguintes características:

- As funcionalidades são independentes umas das outras e podem ser implementadas por um desenvolvedor específico ou um par de desenvolvedores, mas quando já são definidos esses desenvolvedores, eles não podem trocar de funcionalidade durante o desenvolvimento.
- As funcionalidades não podem ser quebradas em tarefas, ou seja, não é possível que uma determinada funcionalidade seja desenvolvida por mais de dois desenvolvedores (programação em pares).
- O time de desenvolvimento permanece o mesmo durante todo o projeto, bem como as duplas definidas para o desenvolvimento com programação em pares.
- Ao final de cada sessão de desenvolvimento, são calculados a quantidade de linhas de código e os erros produzidos por cada mil linhas de código.

A duração da sessão de desenvolvimento é medida em horas, e para que possa ser mais realista se comparado a projetos reais, elas podem assumir valores aleatórios entre uma e oito horas de sessão de desenvolvimento. Outro fator importante é que as equações que são responsáveis pela regulagem e execução do modelo foram retiradas de modelos já existentes (MELIS, 2006), ou seja, sua veracidade é comprovada.

4.2.1 Entidades do Modelo

Entidade: Funcionalidades

As funcionalidades são os itens presentes no backlog do produto, ou seja, são os requisitos que devem ser implementados pelo time de desenvolvedores durante a sessão de desenvolvimento.

As funcionalidades possuem alguns atributos:

- *estimatedPoints*, que representa o esforço estimado para realização de uma determinada funcionalidade.
- *estimationError*, indicando o erro para cada estimativa feita.
- *targetPoints*, que representa o esforço real necessário para completar a funcionalidade, sendo dada pela fórmula que utiliza a estimativa e o possível erro de estimativa da equipe.
- *pointsDone*, que é o esforço atualizado após cada sessão de desenvolvimento.
- *Classes, methods e locs* que representam a quantidade de classes, métodos e linhas produzidas para finalização da funcionalidade.
- *Defects*, que representa o numero de defeitos injetados durante a implementação.

$$TargetPoints = (1 + EstimationError) \cdot EstimatedPoints \quad (1)$$

A fórmula (1) é utilizada para obter os pontos totais presentes em cada funcionalidade. Ela é dada com o acréscimo de um à estimativa do erro, multiplicado pelos pontos estimados. Esta fórmula faz um cálculo com base na

estimativa da equipe em relação a pontos e ao possível erro de estimativa, retornando a quantidade correta de pontos da funcionalidade que devem ser implementados durante a sessão de desenvolvimento atual.

Como pode ser visto em (MELIS, 2006), os valores para os pontos de estimativa e erro de estimativa podem ser fixos. Porém, para que este trabalho se aproxime mais da realidade de um projeto de uma empresa, esses valores serão definidos dentro de um intervalo padrão também descrito no trabalho citado anteriormente, em (MELIS, 2006), demonstrando que para cada funcionalidade diferente a equipe pode fazer estimativas diferentes.

Segundo (MELIS, 2006), os valores para os pontos de estimativa variam de 12 até 15 pontos, e os valores de erro de estimativa variam de 0.9 até 2.0.

4.2.2 Atores do Modelo

Durante o processo as atividades devem ser executadas por um ou mais atores. Os atores do simulador são os desenvolvedores que juntos formam o time de desenvolvedores.

Ator: Time

O time pode ser considerado um ator, pois é formado por um grupo de desenvolvedores, além do fato de que o time realiza ações que interferem nas atividades e modificam atributos do modelo de simulação.

O time é composto por uma coleção de desenvolvedores, que é um dos atributos desse ator. Um outro atributo é a velocidade do time, que é dada como um parâmetro de entrada do modelo. A velocidade inicial do time representa a velocidade em que o time pode implementar um projeto de software. Esse atributo é atualizado durante o desenvolvimento, pois é totalmente relacionado com a velocidade individual de cada desenvolvedor. A velocidade do time sofre uma variação no decorrer do projeto e a atualização desse atributo é dada pela seguinte fórmula, descrita em (MELIS, 2006):

$$V_{team}(t_i) = \sum_j V_{D_j}(t_i) \quad (2)$$

A velocidade do time no instante ti é dada pelo somatório da velocidade de cada um dos desenvolvedores nesse mesmo instante, ou seja, é a velocidade atual do desenvolvedor, que pode aumentar de acordo com o aumento de suas habilidades.

Ator: Desenvolvedor

Os desenvolvedores são o ponto principal do simulador, pois eles são responsáveis por todas as ações que ocorrem durante a sessão de desenvolvimento. Sendo o ponto central, sabe-se que assim como em projetos reais o sucesso ou não do projeto depende dos desenvolvedores. Os desenvolvedores possuem alguns atributos: experiência, habilidade, habilidade máxima e velocidade inicial.

Todo desenvolvedor começa com experiência zero e este atributo vai aumentando de acordo com o tempo de desenvolvimento no projeto. Os atributos habilidade, habilidade máxima e habilidade inicial são iniciados de forma aleatória e individual para cada desenvolvedor. A habilidade máxima, é um valor fixo entre sete e dez, que representa o nível máximo de habilidade que um desenvolvedor pode ter durante o projeto. A velocidade inicial do desenvolvedor é dada pelo quociente entre a velocidade inicial do time e o numero total de desenvolvedores.

A experiência é o trabalho acumulado nos dias de sessão de desenvolvimento no projeto e é atualizado no final de cada uma das sessões de desenvolvimento. A equação (3), descrita em (MELIS, 2006), é responsável por atualizar este atributo:

$$E(t_i) = E(t_{i-1}) + \Delta t_{sess}(t_i) \quad (3)$$

A nova experiência do desenvolvedor é dada pela soma da experiência que ele possuía até o instante anterior ao atual, somado com o tempo total de desenvolvimento daquela sessão especifica.

A habilidade pode ser descrita como a facilidade ou conjunto de qualidades que o desenvolvedor tem para o desenvolvimento de um projeto de software. E está numa escala de no mínimo um e no máximo dez. Inicialmente os desenvolvedores recebem um valor aleatório entre um e três para sua habilidade

inicial. A habilidade do desenvolvedor é atualizada também após cada sessão de desenvolvimento, onde o coeficiente de aprendizado influencia diretamente no aumento da habilidade dos desenvolvedores. É dada pela equação (4), descrita em (MELIS, 2006):

$$skill(t_i) = skill(t_{i-1}) + \Delta E(t_i) \cdot K_{learn} \quad (4)$$

A nova habilidade do desenvolvedor é dada pela soma de sua habilidade no instante anterior ao atual, somado com sua experiência atual multiplicada pelo coeficiente de aprendizado, o qual pode variar de acordo com a escolha das práticas de desenvolvimento.

Por fim, todo desenvolvedor possui uma habilidade máxima, que representa qual o valor máximo de habilidade esse desenvolvedor pode atingir. Esse valor está entre sete e dez pontos.

Com base em resultados empíricos sabe-se que coeficiente de aprendizado é de suma importância no aprendizado dos desenvolvedores, mas esse coeficiente pode variar de acordo com a técnica utilizada. Estima-se que a programação em pares aumenta em 8% a habilidade do desenvolvedor (VIVEKANANDAN, 2004).

Existem dois valores diferentes para o coeficiente de aprendizado dos desenvolvedores, que estão descritos em (MELIS, 2006):

$$K_{learn} = \begin{cases} 0.009 & \textit{Programação Solo} \\ 0.00972 & \textit{Programação em Pares} \end{cases}$$

Figura 3 - Coeficiente de Aprendizado em Relação às Práticas de Programação

Todo desenvolvedor possui uma velocidade de desenvolvimento que descreve o quão rápido esse desenvolvedor pode completar o desenvolvimento de uma Funcionalidade por dia de sessão de desenvolvimento. Essa velocidade de

desenvolvimento é atualizada de acordo com o incremento de sua habilidade e é descrita pela equação (5), descrita em (MELIS, 2006):

$$V_D(t_i) = V_D\,|_{init} + log[Skill(t_i)] \quad (5)$$

A equação (5) calcula a velocidade atual de desenvolvimento de cada desenvolvedor, e é dada pela soma de sua velocidade inicial somada com o nível atual de sua habilidade em forma de log.

Em relação à disponibilidade do desenvolvedor, neste projeto os desenvolvedores possuem um atributo que define se eles estão disponíveis ou não para desenvolvimento. Desse modo, duas possibilidades existem. Quando se deseja programação em pares é necessário ter dois desenvolvedores disponíveis para iniciar o desenvolvimento de uma nova funcionalidade. Quando utiliza a programação solo, é necessária somente a disponibilização de um desenvolvedor.

4.2.3 Atividades do Modelo

Como já foi descrito anteriormente, o trabalho não é totalmente baseado no scrum, pois esse processo possui muitas características quando empregado em um projeto, além das sessões de desenvolvimento ou cálculo do tamanho em horas das sessões de desenvolvimento. Algumas dessas características são reuniões de planejamento, reuniões diárias e sessões de desenvolvimento.

O objetivo principal deste trabalho é avaliar qual conjunto de práticas de desenvolvimento melhor se adequa ao framework scrum, tendo como foco principal a sessão de desenvolvimento. Para empregar os outros componentes do scrum (reuniões e planejamentos), levamos em conta somente o tamanho do sprint, a quantidade de sprints e a quantidade de funcionalidades presentes no backlog do produto, sendo que todos esses são dados de entrada para o simulador.

Tendo em mãos todos esses dados é possível derivar o que é necessário para iniciar a sessão de desenvolvimento, que é vista como única atividade do modelo.

Atividade: Sessão de Desenvolvimento

Durante a sessão de desenvolvimento os desenvolvedores implementam o que foi levantado como requisitos e inseridos no backlog do produto como sendo uma funcionalidade. Para tal atividade é necessário saber o tempo de cada sessão de desenvolvimento. Buscando uma maior realidade se comparado aos projetos reais, utiliza-se a equação (6) para calcular a quantidade de horas de sessão de desenvolvimento de todo o projeto:

TamHorasSess = (8hs*TamSprint*NumSprints) – ((16hs*NumSprints) + 15min*(TamSprint-2)*NumSprints) **(6)**

Para compreender a seguinte equação, é bom exemplificar que leva-se em conta um trabalho diário de oito horas, ou seja, durante oito horas diárias os desenvolvedores e os outros componentes do projeto estão presentes na empresa para realizar as atividades como reuniões e sessões de desenvolvimento. Sendo assim, a fórmula acima descrita pode ser dividida em três partes:

- Parte 1: **(8hs*TamSprint*NumSprints) –** Representa as oito horas diárias de trabalho, multiplicadas pelo tamanho do sprint em dias e o número de sprints que serão realizados. Esta parte dá a quantidade total de horas que o projeto possui.
- Parte 2: **((16hs*NumSprints) –** Representa as reuniões de planejamento e revisão que são realizadas no início e no fim de cada um dos sprints a serem realizados. O valor desta parte da equação deve ser subtraído do total de horas do projeto, obtido anteriormente.
- Parte 3: **15min*(TamSprint-2)*NumSprints –** Representa os 15 minutos de reuniões diárias que acontecem todos os dias, exceto no primeiro e último dia do projeto.

Ao final pode-se obter a quantidade de horas que será reservada a sessão de desenvolvimento durante todo o projeto. Portanto temos:

TamHorasSess = Parte 1 – (Parte 2+Parte 3)

Após obter a quantidade de horas, através da equação (6), no próximo passo dividiremos essa quantidade de horas de sessão de desenvolvimento em cada um

dos dias do projeto. As sessões podem variar de uma até oito horas de desenvolvimento diário, variando de acordo com o projeto, pois essa distribuição é feita de forma aleatória.

Outra característica importante da sessão de desenvolvimento é que ela pode ser realizada em quatro cenários diferentes, que são:

Cenário um: Programação Solo e Sem DDT.

Cenário dois: Programação Solo e Com DDT.

Cenário três: Programação em Pares e Sem DDT.

Cenário quatro: Programação em Pares e Com DDT.

Sabe-se que cada um desses cenários influencia diretamente na sessão desenvolvimento, pois cada um deles atribui valores diferentes em algumas características dos desenvolvedores, bem com influencia na qualidade da implementação produzida.

Produção de Código-Fonte: É esperado que em cada sessão de desenvolvimento seja produzido novo código, referente aos requisitos do projeto. Obviamente que existe uma injeção de defeitos nesse modelo, que é diretamente influenciada por diferentes práticas, como por exemplo, o uso ou não de DDT. O código-fonte é medido através da quantidade de linhas de código, métodos e classes produzidas e que são dadas pela seguinte equação, descrita em (MELIS, 2006):

$$\Delta S_i(t_j) = \Delta t_{sess}(t_j) \cdot P_i(t_j) \qquad i = C, M, L \qquad (7)$$

A equação (7) é dada pela multiplicação entre o tempo da sessão de desenvolvimento juntamente com a produtividade esperada para a implementação de linhas de código, métodos e classes no projeto. A equação se mantém a mesma, modificando apenas o valor da produtividade esperada, que é diferente em relação a linhas de código, métodos e classes no projeto.

Os valores descritos como C, M e L são dados de entrada do simulador e representam a produtividade da equipe em termos de linhas de código, métodos e

classes provenientes de projetos anteriores. Ao final dessas sessões de desenvolvimento esses valores de linhas de código, métodos e classes são atribuídos a suas respectivas Funcionalidades, para que seja possível saber o quão foi necessário produzir em termos de código-fonte para completá-la.

Cálculo de Defeitos no Modelo: O número de defeitos presentes no modelo é dado pela equação (8), descrita em (MELIS, 2006):

$$\Delta D(t_j) = \Delta S_L(t_j) \cdot d(t_j) \tag{8}$$

A equação (8) é dada pela multiplicação da quantidade de linhas de código produzidas naquela sessão atual de desenvolvimento multiplicada pelo valor de defeitos injetados no modelo.

Pode ser visto em diversos trabalhos que diferentes valores para cálculo de defeitos são utilizados para estimar erros no desenvolvimento. Porém, sabe-se que esses valores variam, pois estão diretamente relacionados à habilidade de cada desenvolvedor, supondo-se que desenvolvedores mais habilidosos errem menos em comparação com desenvolvedores menos habilidosos. Neste trabalho quando a programação é feita em pares, a predominância é do desenvolvedor com maior habilidade, pois estima-se que ele irá instruir o outro desenvolvedor menos experiente, melhorando assim a qualidade do desenvolvimento. Em (MELIS, 2006) podemos encontrar a seguinte tabela de valores para cálculo de defeitos relacionada com a habilidade do desenvolvedor:

Tabela 1 – Cálculo de Defeitos por nível de habilidade

Habilidade do Desenvolvedor	Defeitos Injetados no Modelo
1-3	1.32
4-7	1.00
8-10	0.76

Porém, os valores de defeitos ainda podem diminuir, devido às práticas de programação escolhidas. Sabe-se que a programação em pares produz uma melhor qualidade do que a programação solo, pois o desenvolvedor mais habilidoso auxiliará o menos habilidoso, de modo que ambos produzam um código de melhor qualidade. Em (MELIS, 2006) os parâmetros ajustados definem que com a utilização de programação em pares, os defeitos injetados diminuem em 15% no desenvolvimento. Sendo assim, quando a programação em pares é escolhida, o simulador emprega uma redução de 0.15 nos defeitos injetados no modelo.

Do mesmo modo em (LAURIE, MICHAEL, MADLEN, 2003) e (YNCHAUSTI, 2001) são feitos estudos onde mostram-se que a utilização de DDT faz com que os defeitos injetados caiam mais ainda, chegando a diminuição de 40%, se comparado com projetos que utilizam técnicas de desenvolvimentos mais tradicionais. Sendo assim, quando o DDT é escolhido, o simulador emprega uma redução de 0.40 nos defeitos injetados no modelo.

4.3 Kernel do Simulador

O Simulador é composto por uma classe principal, esta que contém sua interface e toda lógica necessária para a correta execução da simulação.

Principal: A rotina principal contém o método "run()" que dispara a execução dos eventos no simulador. Pelo fato dos eventos serem ligados à sessão de desenvolvimento, durante a mesma, outras ações são executadas para completar a simulação total do projeto.

MatchEngine: Esta classe contém toda a lógica relacionada a sessão de desenvolvimento. Através de uma instância dessa classe criada na classe principal, que todas as ações são feitas em relação ao eventos.

Está classe contém todos os métodos correspondentes às fórmulas descritas nas sessões anteriores. Para uma melhor compreensão do modelo, basta analisar a Figura 4, que é o diagrama de classes do simulador implementado.

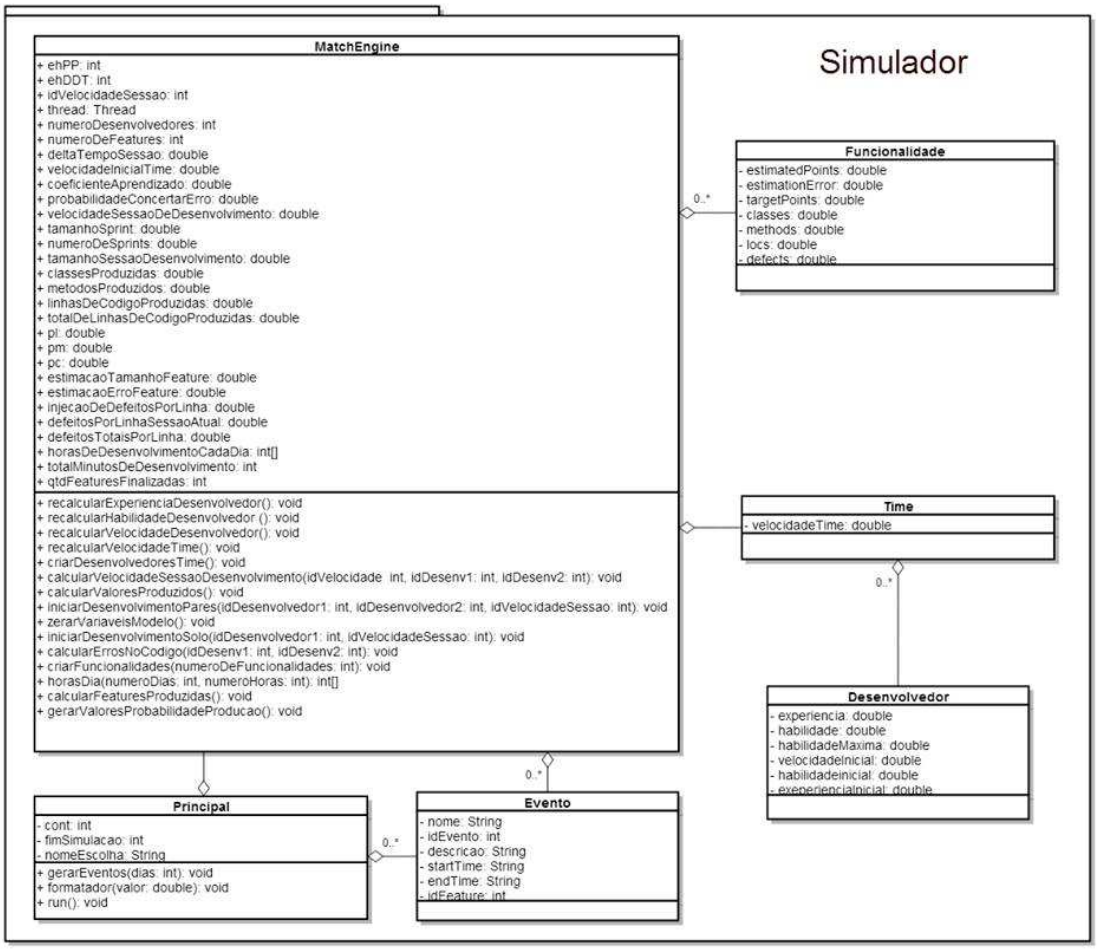

Figura 4 - Diagrama de Classes do Modelo

4.4 Fluxo de Simulação

Para um melhor entendimento do funcionamento do simulador, é necessário esclarecer alguns pontos e especificar as atividades que são executadas ao iniciar o processo de simulação do processo de desenvolvimento de software utilizando scrum.

Inicialmente é necessário fornecer os parâmetros de entrada do simulador que são os seguintes: número de funcionalidades, número de desenvolvedores, velocidade Inicial do time, tamanho do sprint em dias, número de sprints, produtividade em linhas de código, produtividade em métodos e produtividade

em classes. O usuário também deve selecionar um dos quatro cenários descritos na sessão **4.2.3**.

Após isso são criados de forma totalmente aleatória todos os desenvolvedores e todas as suas características básicas. Com os parâmetros inseridos de forma correta a simulação pode ser iniciada e de inicio o simulador irá calcular quantas horas serão reservadas para sessões de desenvolvimento diárias, utilizando uma distribuição aleatória de horas durante os dias de projeto.

Após isso se dará inicio à sessão de desenvolvimento daquele dia e de todos os dias posteriores, lembrando que todos os cálculos das fórmulas apresentados nas sessões anteriores são executados em cada sessão de desenvolvimento.

Para cálculo de quantas funcionalidades são produzidas, como pode ser visto em (MELIS, 2006) um ponto da funcionalidade tem estimativa de ser finalizado com 30 minutos de trabalho. Com intuito de aumentar a realidade deste trabalho, resolvemos nos basear também nas práticas de programação em pares e DDT para estimar o tempo de trabalho de cada ponto das funcionalidades. Isto pode ser visto na tabela 2:

Tabela 2 - Minutos de acordo com as praticas escolhidas

Técnica de Programação	Tempo de Implementação em Minutos das Funcionalidades
Programação em Pares e com DDT	90 minutos
Programação em Pares e sem DDT	30 minutos
Programação Solo e com DDT	60 minutos
Programação Solo e sem DDT	60 minutos

É importante ressaltar a versatilidade do simulador, pois ele pode fornecer quatro diferentes cenários para uma comparação em relação às possibilidades do projeto atual, como por exemplo, possuir muitos desenvolvedores e dividi-los em pares e também possuir especialistas em DDT para trabalhar no desenvolvimento do projeto.

Todos os parâmetros tem grande influencia no simulador, portanto, em alguns casos pode ocorrer a falha do projeto, onde não seria possível finalizar a implementação das funcionalidades presentes no backlog do produto no intervalo de dias programados.

Ao final da execução serão obtidos alguns resultados que se referem a: quantidade de dias do projeto, quantidade de linhas de código produzidas, quantidade de funcionalidades implementadas e quantidade de erros por linhas de código. Com esse resultado em mãos é possível fazer uma avaliação do projeto e buscar novas alternativas (novos parâmetros) que possam melhorar o desempenho da equipe no projeto.

4.5 Calibração e Validação do Modelo

Quando criam-se simuladores de processos de desenvolvimento de software que buscam se aproximar da real execução de projetos, um dos principais problemas acaba sendo a sua calibração e validação. A busca pela simulação vem do fato de que é praticamente impossível fazer testes em projetos reais, pois os mesmos gastam tempo, dinheiro e muitos outros recursos que se utilizados em excesso podem causar graves prejuízos às empresas de desenvolvimento de software.

Quando simuladores são implementados deve-se obter dados de projetos em empresas reais para ter uma base de comparação, que mostra o quão próximo sua simulação se aproxima da realidade. A obtenção desses dados é complexa por diversos fatores como, por exemplo, o fato de que as empresas desenvolvem seus projetos de forma privada, sem acesso ou interferência externa. Outro problema é que seria muito difícil encontrar empresas que utilizam desenvolvimento de software seguindo à risca as práticas provenientes do SCRUM mescladas com uso de Programação em Pares e DDT.

Para tal calibração, utilizamos o trabalho de (MELIS, 2006), porém com algumas adaptações, pois o objetivo do trabalho é a análise do impacto das práticas de programação no scrum, ou seja, é focado somente na sessão de desenvolvimento e não nas práticas XP, como foi descrito no trabalho. Com os seguintes parâmetros de entrada, obtivemos as seguintes saídas:

Tabela 3 - Descrição e valores dos parâmetros de entrada.

Parâmetros de Entrada	Valor do Parâmetro
Número de Funcionalidades	27
Número de Desenvolvedores	4
Estimativa de Pontos	12-15
Erro de Estimativa	0.9-2.0
Velocidade Inicial do Time	16
Produtividade de Linhas de Código	80-156
Produtividade de Métodos	3-7
Produtividade de Classes	0.3-1.8

Para esses valores de entrada as saídas foram:

Tabela 4 - Saída relacionada aos parâmetros de entrada

Variáveis de Saída	Simulação	Simulação (MELIS, 2006)
Dias de Projeto	60	61.9
Funcionalidades finalizadas	27	28.2
Linhas de Código [KLoc]	8.3	9.9

Obviamente que para cada vez que o simulador é executado, as saídas são diferentes devido à sua similaridade com projetos reais. Mas o objetivo da calibragem é validar o modelo obtendo-se resultados sempre próximos do correto.

Em (MELIS, 2006) o foco é grande em todas as partes do XP, diferentemente do nosso trabalho que visou abordar somente a sessão de desenvolvimento descrita nessa Tese e adicionar conceitos do scrum juntamente com programação em pares e DDT. Sendo assim, foram avaliados somente os resultados pertinentes à similaridade do nosso modelo com o descrito na citação anterior.

5
Resultados

O simulador foi desenvolvido com intuito de avaliar qual conjunto de práticas de desenvolvimento melhor se adequa ao framework scrum, buscando demonstrar as vantagens e desvantagens do uso ou não de cada prática.

Os testes foram baseados nos quatro casos descritos em **4.2.3**, e em todos os casos possuem os mesmos parâmetros de entrada do simulador. Os objetivos dos testes foram observar a influência do uso ou não da programação em pares e de DDT, verificando quais as diferenças nas execuções do simulador, quando os casos são testados.

As tabelas 5,6,7,8 e 9 nos mostram as entradas e saidas respectivas para cada um dos casos teste que foram inseridos no simulador. A simulação foi executada 50 vezes para cada um dos quatro casos possíveis. A ferramenta possui caixas de marcação onde é possível escolher quais práticas serão utilizadas. Também possui caixas de texto onde devem ser preenchidas as informações principais de entrada do simulador: número de funcionalidades, número de desenovolvedores, velocidade inicial do time, tamanho do sprint em dias e o número de sprints.

O resultado nos mostra em quantos dias o projeto foi realizado, quantas funcionalidades foram implementadas, quantas linhas de código foram implementadas em média, quantos defeitos ocorreram para cada mil linhas de código em média, qual o desvio padrão dos valores obtidos nas linhas de código produzidas e o desvio padrão obtido nos erros pora cada mil linhas de código.

Tabela 5 - Parâmetros de Entrada

Variáveis de Entrada	Valor da Variável
Número de Funcionalidades	131
Número de Desenvolvedores	6
Velocidade Inicial do Time	28
Tamanho do Sprint (Dias Úteis)	33
Número de Sprints	6

Tabela 6 - Resultado para Programação em Pares e DDT

Variáveis de Saída	Valor da Variável
Dias de Projeto	150 198 250
Funcionalidades Implementadas	131
KLocs (Média)	49.95
Defeitos/KLocs (Média)	69.02
KLocs (Desvio Padrão)	9.01
Defeitos/KLocs (Desvio Padrão)	0.04

Tabela 7 - Resultado para Programação em Pares e Sem DDT

Variáveis de Saída	Valor da Variável
Dias de Projeto	198
Funcionalidades Implementadas	131
KLocs (Média)	48.89
Defeitos/KLocs (Média)	128.94
KLocs (Desvio Padrão)	8.11
Defeitos/KLocs (Desvio Padrão)	0.31

Tabela 8 - Resultado para Programação Solo e DDT

Variáveis de Saída	Valor da Variável
Dias de Projeto	198
Funcionalidades Implementadas	131
KLocs (Média)	92.96
Defeitos/KLocs (Média)	90.07
KLocs (Desvio Padrão)	16.80
Defeitos/KLocs (Desvio Padrão)	0.04

Tabela 9 - Resultado para Programação Solo e sem DDT

Variáveis de Saída	Valor da Variável
Dias de Projeto	198
Funcionalidades Implementadas	131
KLocs (Média)	94.65
Defeitos/KLocs (Média)	150.12
KLocs (Desvio Padrão)	18.80
Defeitos/KLocs (Desvio Padrão)	0.05

Fazendo uma observação nos quatro casos e comparando os resultados, podemos tirar diversas conclusões. Primeiramente, percebe-se que os modelos de programação quando utilizam programação em pares, mesmo que usem DDT ou não, acabam produzindo um número de linhas de código mais baixo se comparado com a programação solo.

Em outra análise, podemos perceber que em ambos os projetos que não utilizam DDT, o número de erros é exageradamente maior do que os que utilizam. Em sessões anteriores dessa Monografia, quando explicávamos a influência do DDT, essa questão foi dita e agora comprovada pelos resultados do simulador.

Analisando também, percebemos que a programação em pares juntamente com o uso de DDT, gera uma grande melhora na qualidade de código, reduzindo assim o número de erros. Sabendo é claro, que o não uso das duas técnicas causa um número de erros muito maior no código implementado.

Uma última análise dos resultados, nos mostra que a alternância entre o caso 2: uso de programação em pares e não uso de DDT, e o caso 3: uso de programação solo e DDT, nos traz resultados semelhantes, porém, deve-se notar que o número de erros diminui quando no caso 3, não é utilizado programação em pares juntamente com o DDT.

Em relação às funcionalidades, como o planejamento recebeu valores de estimativas próximos do real, em ambos os casos todas as funcionalidades foram finalizadas. Porém, é claro, em alguns casos as sessões de revisão e manutenção de código terão de ser maiores, devido ao maior número de erros.

6
Conclusão

A ferramenta desenvolvida através do estudo do scrum e das técnicas de programação em pares e DDT oferecem uma grande facilidade e praticidade quando se necessita simular uma situação real de um projeto. A forma como o simulador avalia as sessões de desenvolvimento, produções de código e erros nas linhas de código fazem com que a sua utilização possa facilitar muito a vida de um gerente de projetos ou até mesmo facilite o aprendizado, mostrando em simulação as vantagens do uso de programação em pares e DDT.

A existência de uma crise do software não foi um consenso entre os profissionais da área. Mas para tornar possível a solução dos problemas enfrentados, foi criada uma nova metodologia com uma abordagem bem diferenciada, que são os métodos ágeis.

O problema em se dirigir um projeto desde sua fase de planejamento até seu término, pode gerar muitas incertezas em uma empresa. O fato de não saber o que se esperar de um projeto, causa grandes problemas. As simulações são criadas exatamente para atacar esse problema, pois não existem possibilidades de realização de testes em projetos reais.

O uso das técnicas de scrum, juntamente com programação em pares e DDT, fez com que o simulador pudesse mostrar de forma muito clara a função destas duas técnicas. Primeiramente o DDT tem como objetivo obter um código “limpo”, com qualidade e que funcione. A programação em pares é uma forma eficaz de reduzir a incidência de erros em um sistema. Isso se deve em grande parte às visões complementares que atuam durante o uso dessa prática. Quando dois desenvolvedores estão programando em par, um deles está programando. O outro está sentado ao lado, olhando para a mesma tela e preocupado em resolver o mesmo problema. Ambos estão trabalhando juntos na solução. Eles conversam o tempo todo e trocam ideias sobre a solução, melhorando assim o nível do código.

6.1 Trabalhos Futuros

À medida que é possível detalhar mais ainda todas as técnicas que estão por trás do scrum, é possível abranger mais ainda esse método ágil, nos levando a uma realidade ainda maior se comparado com projetos reais. Neste trabalho, as reuniões não foram simuladas, pois partimos do pressuposto de já possuir um backlog do produto pré-definido, restando apenas implementar as funcionalidades presentes no mesmo.

Alguns exemplos de adições à simulação seriam simular também:

- Reuniões de Planejamento da Sprint
- Reuniões Diárias
- Revisão da Sprint
- Retrospectiva da Sprint

Com essas novas aquisições, grandes mudanças devem ser feitas no simulador, pois seria necessário adicionar diversas fórmulas que pudessem nos dar os valores reais que seriam produzidos em cada uma dessas reuniões durante o projeto. Portanto, um trabalho futuro com essas aquisições, tornaria a ferramenta cada vez mais completa e próxima dos projetos reais.

7

Referências Bibliográficas

Alexis, O; Jürgen, M; *Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes,* Fraunhofer Institute for Experimental Software Engineering, Fraunhofer-Platz 1, 67663, Kaiserslautern, Germany, 2006.

Alistair, C; Laurie, W. *The costs and benefits of pair programming*. In Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000), Cagliari, Sardinia, Italy, June 2000.

Baker, A; Navarro, E; Hoek, A. *An experimental card game for teaching software engineering process*, The Journal of System and Sofwate, Elsevier, 2005.

Ball, P. University of Strathclyde. *Introduction to discrete event simulation*. Published on: http://www.dmem.strath.ac.uk/pball/simulation/simulate.html, 2001.

Boby, G; Laurie, W. *An initial investigation of test driven development in industry*. In Proceedings of the 2003 ACM symposium on Applied computing, p.1135–1139. ACM Press, 2003.

Crispen, L; House, L. *Testing Extreme Programming*. Addison Wesley, ma: addison wesley pearson education edition, 2003.

Erdogmus, H; Morisio, M; Torchiano M. *On the effectiveness of the test-first approach to programming*. IEEE Transactions on Software Engineering, 31(3):226–237, March 2005.

Fishman, G. S; *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, Berlin, 2001.

Craig, D; *Extensible hierarchical object-oriented logic simulation with an adaptable graphical user interface*. Master of science, School of Graduate Studies,

Department of Computer Science, Memorial University of Newfoundland, 1996.

Hanna, H; Pekka, A. *A multiple case study on the impact of pair programming on product quality*. In ICSE '05: Proceedings of the 27th international conference on Software engineering, pages 495–504, New York, NY, USA, 2005. ACM Press.

Haumer, P. *Overview to Eclipse Framework Composer,* Eclipse Foundation, 2007.

Henderson-Sellers, B; Gonzalez-Perez, C. *A comparsion of four process metamodels and the creation of a new generic standard*, Univesity of Technology, Sidney, Australia. Publicado em: Information and Software Technology nº 47, , p.49-65, 2005.

John, T; Nosek. *The case for collaborative programming*. *Commun*. ACM, 41(3): 105–08, 1998.

Jonathan, E; Alexander, L. *Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Mode,* University of Colorado Department of Computer Science Technical Report CU-CS-840-97 May 1997.

Kellner, M; Madachy, R; Raffo, D. *Software process simulation modeling: Why? What? How?*, The Journal of Systems and Software, 46(2–3):91–105, April 1999.

Kreutzer, W. *System Simulation - Programming Styles and Languages*. Addison Wesley, Reading (U.S.A.), 1986.

Laurie, W; *at al*. *Strengthening the case for pair programming*. IEEE Software, 17(3):19–25, July/August 2000.

Laurie, W; Michael, M; Vouk, M. *Test-driven development as a defect-eduction practice*. In Proceedings of the 14th Symposium on Software Reliability Engineering (ISSRE'03), p.34–45, 2003.

Laurie, W; Robert, R. *Pair Programming Illuminated*, chapter 4: vercoming Management Resistance to Pair Programming, pages 33–44. Addison Wesley Longman Publishing Co., Inc., 2002.

Marcos, Melis. *A Software Process Simulation Model of Extreme Programming*. Dissertação de Doutorado. Instituição, Local, February 7, 2006.

Navarro, E. *SimSE: A Software Engineering Simulation Environment for Software Process Education* , Universidade da Califórnia, Irvine, 2006.

OMG, *Software & System Process Engineering Metamodel Specification*, Versão 2.0, 2008.

Paula Filho, W. *Engenharia de Software, métodos e padrões.* LTC, 2009.

Pfleeger, S. *Engenharia de Software: Teoria e Prática*, Prentice Hall, 2004.

Pressman, R. *Engenharia de Software.* Pearson Makron Books, 1995.

Ramsin, R., Paige, F. *Process-Centered Review of Object Oriented Software Development Methods*, ACM Computing Survey, Vol 40, N 1, artigo 3, 2008.

Rus, I; Halling, M; Biffl, S. *Supporting decisionmaking in software engineering with process simulation and empirical studies*. International Journal of Software Engineering and Knowledge Engineering, 13(5):531–545, 2003.

Schriber, T. J; Daniel T. Brunner, D. J. *Inside discrete-event simulation software: how it works and why it matters*. In WSC '99: Proceedings of the 31st conference on Winter simulation, pages 72–80. ACM Press, 1999.

Schwaber, K; Sutherland, J; *Guia do Scrum*. scrum.org. 2011.

Sommerville, I, *Engenharia de Software*, Oitava Edição, Pearson Addison-Wesley, 2007.

Tomás, M; Félix, G; Mario, P. Towards A SPEM v2.0 *Extension to Define Process Lines Variability Mechanisms, 2011.*

Vivekanandan, K. *The Effects of Extreme Programming on Productivity, Cost of Change and Learning Efficiency*. PhD thesis, Department of Computer Science, Ramanaujam School of Mathematics and Computer Sciences, Pondicherry University, India., 2004.

Ynchausti, R. A. *Integrating unit testing into a software development teams process.* Published on: http://www.agilealliance.org/articles/ynchaustirandyaintegr/file, 2001.