UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Guilherme Schoepping

UM ESTUDO EXPLORATÓRIO A PARTIR DE UM *FRAMEWORK* PARA SELEÇÃO DE PRÁTICAS ÁGEIS

Documento submetido ao Programa de Pós-Graduação em Ciência da Computação como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora:
Patrícia Vilain

Florianópolis, 2012

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Schoepping, Guilherme
UM ESTUDO EXPLORATÓRIO A PARTIR DE UM FRAMEWORK PARA SELEÇÃO DE PRÁTICAS ÁGEIS [dissertação] / Guilherme Schoepping ; orientadora, Patrícia Vilain - Florianópolis, SC, 2012.
123 p. ; 21cm

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Métodos ágeis. 3. Práticas ágeis. I. Vilain, Patrícia. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Guilherme Schoepping

UM ESTUDO EXPLORATÓRIO
A PARTIR DE UM *FRAMEWORK*
PARA SELEÇÃO DE PRÁTICAS ÁGEIS

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Ciência da Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação

Local, 29 de outubro de 2012.

_____
Prof. Ronaldo dos Santos Mello, Dr.
Coordenador do Curso - USP

Banca Examinadora:

_____
Prof.ª Patrícia Vilain, Dr.ª
Orientadora - UFSC

_____
Prof. Raul Sidnei Wazlawick, Dr.
UFSC

_____
Prof. Ricardo Pereira e Silva, Dr.
UFSC

_____
Prof. Alfredo Goldman vel Lejbman, Dr.
USP

AGRADECIMENTOS

Agradeço aos meus pais pelos constantes esforços dedicados a mim, a minha noiva por tamanho amor e apoio, a minha orientadora pela incansável paciência e dedicação, ao Programa de Pós-Graduação em Ciência da Computação pela oportunidade, e a todos que indiretamente contribuíram para que eu continuasse e concluísse este trabalho.

There's a difference between knowing the path and walking the path.

Morpheus, The Matrix, 1999

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O principal objetivo dos métodos ágeis existentes é promover o desenvolvimento eficiente de software através de práticas que priorizam a comunicação com o cliente e entregas frequentes. Cada método ágil apresenta um conjunto próprio de práticas. Com esta diversidade de práticas torna-se interessante a construção de novos processos ágeis que contemplem apenas as práticas mais adequadas a partir destes métodos. O problema, entretanto, é que a combinação de práticas de diferentes métodos ágeis não garante, necessariamente, que o novo processo definido seja ágil. Este trabalho avalia a agilidade do conjunto de práticas de um *framework* de práticas ágeis e busca identificar quais práticas apresentam maior harmonia quando usadas no mesmo processo. A agilidade das práticas é avaliada através dos dados de uma grande pesquisa de opinião online e a harmonia entre elas é identificada através da técnica de análise de agrupamentos. Os melhores resultados foram apresentados pelas práticas de Integração contínua, Desenvolvimento lado a lado e Testes de aceitação. A análise de agrupamentos, por sua vez, formou quatro grupos de práticas: o primeiro formado por Projeto da arquitetura do sistema e Lista de requisitos; o segundo por Desenvolvimento coletivo de código, Integração contínua, Refatoração e Testes de aceitação; o terceiro por Projeto da iteração e Modelagem geral; e o quarto por Desenvolvimento lado a lado e Reuniões diárias.

Palavras-chave: Métodos Ágeis, Práticas Ágeis, Scrum, Extreme Programming, Feature Driven Development, Refatoração, Programação em pares, Testes de aceitação.

ABSTRACT

The main objective of agile methods is to promote efficient software development through practices that prioritize communication with the client and frequent deliveries. Each agile method presents its unique set of practices. This diversity of practices may lead to the definition of new agile processes that include only the more appropriate practices from these methods. The problem, however, is that combining practices from different methods does not guarantee that the resulting process can be considered agile. This work assesses the agility of a set of practices of a *framework* for selecting agile practices and seeks to identify which practices provide more harmony when used in the same process. The agility of the practices is evaluated using data from a large online survey and the harmony between them is identified by the technique of cluster analysis. The best results were presented by the practices of Continuous integration, Side by side development and Acceptance tests. The cluster analysis resulted in four practice groups: the first with System architectural design and Requirements list; the second with Collective code ownership, Continuous integration, Refactoring and Acceptance tests; the third with Iteration design and General modeling; and the fourth with Side by side development and Daily meetings.

Keywords: Agile Methods, Agile Practices, Scrum, Extreme Programming, Feature Driven Development, Refactoring, Pair Programming, Acceptance Tests.

1 INTRODUÇÃO

Os softwares têm um importante e fundamental papel no mundo moderno, e seu rápido desenvolvimento e correto funcionamento são uma grande vantagem de negócio. A metodologia ágil se apresenta como uma importante alternativa para o desenvolvimento de sistemas. Seu surgimento tem como finalidade tentar transpor os obstáculos que a engenharia de software convencional enfrenta: atrasos na entrega de software, dificuldade na mudança de requisitos e pouca comunicação entre o cliente e a equipe de desenvolvimento [Pressman 2006].

1.1 Agilidade

O termo agilidade, no contexto de desenvolvimento de software, tornou-se abrangente a partir do Manifesto Ágil [Beck 2001], um trabalho realizado por um grupo de especialistas na área de software. Embora esse manifesto não forneça uma definição precisa sobre agilidade, seu conteúdo é a principal referência para a filosofia e valores do movimento ágil. O manifesto fala da necessidade de se ter uma equipe qualificada e com boa comunicação entre seus membros para alcançar os objetivos do projeto; destaca também a importância da presença do cliente durante o processo, pois este é quem define o que deve ser feito e pode ajudar a esclarecer eventuais dúvidas. A entrega frequente de software funcionando também é enfatizada, bem como a capacidade da equipe em absorver as mudanças de requisitos durante o desenvolvimento.

Muitas respostas a respeito sobre o que é o desenvolvimento ágil de software são referências aos valores e princípios deste Manifesto. Alguns autores, contudo, tentam definir claramente esse conceito. Scott Ambler [Ambler] define desenvolvimento ágil de software da seguinte maneira:

> *"Uma abordagem iterativa e incremental (evolutiva) de desenvolvimento de software que é executada de maneira altamente colaborativa, por equipes auto-organizadas, com uma governança eficaz, que produzem soluções de alta qualidade de forma rentável e em tempo oportuno para incorporar as necessidades de mudanças dos seus interessados."*

Quatro autores – Pekka Abrahamssom, Outi Salo, Jussi

Ronkainen e Juhani Warsta – realizaram um trabalho de revisão e análise de diversos métodos ágeis [Abrahamssom 2002] e também propuseram uma interessante definição de método ágil:

> *"Um método pode ser considerado ágil quando o desenvolvimento de software for incremental (pequenas entregas de software, com ciclos rápidos), cooperativo (clientes e desenvolvedores trabalham constantemente juntos e com comunicação estreita), direto (o método propriamente é fácil de aprender e modificar, e bem documentado), e adaptativo (capaz de incorporar mudanças de última hora)."*

O desenvolvimento ágil de software é um movimento que segue a filosofia do Manifesto Ágil [Beck 2001], e engloba uma série de métodos. Esses métodos são baseados no desenvolvimento iterativo, onde os requisitos e soluções evoluem através da colaboração de equipes auto-organizadas e multifuncionais [Agile Glossary]. A grande mudança desses novos métodos é a sua abordagem mais flexível para o desenvolvimento de software do que a apresentada pelos métodos tradicionais até então conhecidos. É importante destacar que a agilidade pode ser adicionada em qualquer processo [Pressman 2006] e não é contrária às sólidas práticas de engenharia de software. Uma das principais características de um processo de desenvolvimento ágil é seu comportamento quanto ao acolhimento a mudanças [Jacobson 2002].

Diversos métodos considerados ágeis podem ser encontrados: Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Crystal, Adaptative Software Development (ASD), Dynamic System Development Method (DSDM), entre outros. Naturalmente, existe muita semelhança na abordagem e nas práticas destes diferentes métodos, ainda que hajam características que tornam cada um deles singular.

Diante da variedade de métodos ágeis e das semelhanças e diferenças entre eles, foi proposto um *framework* que auxilia na construção de um novo processo ágil a partir das práticas dos principais métodos existentes [Vilain 2007]. Este *framework* facilita o trabalho de qualquer organização que queira definir seu próprio processo de desenvolvimento ágil, através da seleção de práticas que os envolvidos julguem necessárias. Evita-se, com o uso deste *framework*, a

necessidade de avaliar diferentes métodos ágeis e escolher apenas um deles ou um conjunto de práticas deles. Entretanto, após a utilização do *framework*, surgiram questionamentos sobre a agilidade dos processos formados a partir dele, visto que a variação e combinação das práticas são grandes.

1.2 Objetivos do trabalho

1.2.1 Objetivo geral

O objetivo principal desse trabalho é avaliar a agilidade das práticas do *Framework* de Práticas Ágeis [Vilain 2007] e identificar quais práticas possuem melhor relacionamento em um processo, para fornecer indícios sobre a agilidade das práticas a serem incluídas em um processo de desenvolvimento.

1.2.2 Objetivos específicos

- Utilizar e revisar o *framework* proposto em [Vilain 2007];
- Identificar e avaliar relações entre as práticas do *framework* no que diz respeito à agilidade;
- Mostrar quais conjuntos de práticas possuem maior harmonia para serem usadas em conjunto.

1.3 Justificativas do trabalho

A definição de um novo processo ágil traz como principal qualidade a personalização, pois funcionará de acordo com a filosofia da organização ou projeto. Um novo problema, todavia, pode surgir. Mesmo que cada prática constituinte do novo processo faça parte de um método preexistente e seja considerada ágil, é possível que a combinação de algumas delas diminua o grau de agilidade do novo processo como um todo. É esse o problema que motiva esse trabalho, e a proposta é analisar as práticas do *framework* quanto à agilidade para que essa informação ajude na definição de novos processos a partir do *framework*.

1.4 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma. No capítulo 2 é apresentada uma visão geral do movimento ágil, seus princípios e valores, assim como as características de alguns dos principais métodos ágeis: Scrum, Extreme Programming (XP), Feature Driven

Development (FDD) e Lean Software Development (LSD). Nesse capítulo também é descrita a estrutura do *Framework* de Práticas Ágeis e suas práticas [Vilain 2007].

No capítulo 3 é apresentada a revisão bibliográfica de alguns trabalhos relacionados.

No capítulo 4 é apresentada a abordagem e os critérios utilizados para analisar a agilidade das práticas presentes no *framework* [Vilain 2007].

No capítulo 5 são apresentados dados sobre a realização de uma revisão sistemática da literatura a respeito da efetividade de algumas práticas ágeis que teve como objetivo solidificar os resultados obtidos no capítulo anterior.

Por fim, no capítulo 6, as considerações finais são apresentadas.

2 MÉTODOS ÁGEIS

Historicamente o desenvolvimento de software tem algumas dificuldades comuns, como a satisfação do cliente, a entrega dentro do prazo e o cumprimento do orçamento. Para amenizar estas dificuldades a Engenharia de Software oferece métodos que auxiliam este desenvolvimento. Trata-se de métodos descritivos, ou também chamados de métodos prescritivos, que definem uma lista de atividades a serem seguidas.

Em busca de maior eficiência, nesse cenário, um grupo de 17 desenvolvedores de software publicou um manifesto, que ficou conhecido como Manifesto Ágil [Beck 2001], onde diziam descobrir maneiras melhores de desenvolver software. Neste manifesto algumas características passaram a ser mais valorizadas:

Indivíduos e interações *mais que processos e ferramentas.*
Software funcionando *mais que documentação abrangente.*
Colaboração com cliente *mais que negociação de contratos.*
Resposta à mudança *mais que seguir um plano.*

Seguindo a mesma filosofia, este grupo também publicou uma lista de princípios ágeis para esclarecer e nortear o desenvolvimento ágil.

Os Princípios Ágeis

1. Nossa prioridade mais alta é satisfazer o cliente por meio de entrega adiantada e contínua de software de valor.
2. Acolher modificação de requisitos, mesmo no final do desenvolvimento. Processos ágeis valorizam a modificação para vantagem competitiva do cliente.
3. Entregar software funcionando com frequência (de várias semanas a vários meses), preferencialmente usando uma escala de tempo menor.
4. O pessoal do negócio e os desenvolvedores devem trabalhar juntos diariamente ao longo do projeto.
5. Construir projetos em volta de indivíduos motivados. Dê a eles o ambiente e o apoio que necessitam e confie que eles vão fazer o serviço.
6. O método mais eficiente e efetivo para levar informação de e para uma equipe de desenvolvimento é a conversa face a face.
7. Software funcionando é a principal medida de progresso.
8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem poder

manter um ritmo constante indefinidamente.

9. Atenção contínua para a excelência técnica e para um bom projeto aumenta a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho que não precisou ser feito – é essencial.
11. As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, depois se ajusta e otimiza seu comportamento de acordo.

Hoje, o desenvolvimento ágil de software é formado por um grupo de métodos baseado no desenvolvimento iterativo e incremental. A maioria deles também promove o trabalho em equipe, a colaboração e a adaptabilidade durante o ciclo de desenvolvimento.

Seguindo um caminho histórico, o movimento ágil inspira o surgimento de diversos métodos a cada dia em busca de aceitação pela comunidade de desenvolvimento de software [Pressman 2006]. A seguir, quatro métodos ágeis – Scrum, Extreme Programming, Feature Driven Development e Lean Software Development - são descritos brevemente. É importante notar que eles apresentam muitas semelhanças e seguem, em graus diferentes, os princípios listados acima e ao Manifesto Ágil [Beck 2001].

2.1 Scrum

O Scrum é um *framework* de processo usado no gerenciamento do desenvolvimento de produtos complexos [Schwaber 2011]. Ken Schwaber e Jeff Sutherland apresentaram o Scrum pela primeira vez na conferência OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications*) em 1995. Inicialmente, o *framework* era destinado para projetos de desenvolvimento de software, mas hoje é usado em projetos de diversas áreas.

Sinteticamente, o Scrum consiste em equipes de desenvolvimento (equipe Scrum) associadas com as funções de cada integrante, eventos, artefatos e regras. Cada componente do *framework* tem um propósito específico e é essencial para o uso e sucesso do Scrum [Schwaber 2011]. Baseado no Guia Scrum 2011 [Schwaber 2011], disponibilizado pelos criadores do *framework*, cada componente será apresentado a seguir.

2.1.1 Equipe Scrum

A equipe Scrum é formada pelo *product owner*, a equipe de

desenvolvimento e o *Scrum master*. As equipes possuem duas características: auto-organização e multidisciplinaridade. Auto-organização refere-se à autonomia dos integrantes escolherem como realizar e completar o trabalho, ao invés desta decisão ser tomada por alguém fora da equipe. Equipes multidisciplinares são aquelas onde as competências necessárias para a conclusão do trabalho são encontradas exclusivamente em seus integrantes, sem dependência externa.

O *product owner* é a pessoa responsável por gerenciar o *product backlog*. Isso inclui definir claramente o que deve ser feito, ordenar a prioridade dos itens de trabalho, assegurar que a equipe de desenvolvimento entenda estes itens, entre outras atividades. Para o sucesso do projeto deve existir respeito de toda a organização pelas decisões do *product owner*, pois este é quem decide o que fazer e quando fazer.

A equipe de desenvolvimento é formada pelos profissionais responsáveis pelo desenvolvimento do incremento a ser entregue no final do *sprint*. Como dito anteriormente, são multidisciplinares e auto-organizadas. Todos os seus integrantes são igualmente reconhecidos como desenvolvedores e compartilham a responsabilidade pelo trabalho realizado. No Scrum, a equipe não pode ser dividida em uma unidade menor com o foco a uma determinada área como testes ou análise de negócio. Segundo o Guia [Schwaber 2011], o tamanho da equipe deve ser pequena o suficiente para se manter ágil e grande o suficiente para desenvolver uma trabalho significativo; recomenda-se equipes de 3 a 9 integrantes, pois equipes menores podem encontrar restrições de competência para a conclusão do trabalho e equipes maiores geram muita complexidade para ser gerenciada.

A função do *Scrum master* é assegurar o correto uso e entendimento do Scrum. Isso é alcançado através da cooperação junto a todos os envolvidos. O *Scrum master* pode colaborar com o *product owner* de diversas maneiras: facilitar a comunicação deste com a equipe de desenvolvimento e descobrir novas técnicas para um bom gerenciamento do *product backlog*, por exemplo. Outras atividades do *Scrum master* incluem assessorar a equipe de desenvolvimento e remover impedimentos ao progresso do seu trabalho. Do ponto de vista da organização, o *Scrum master* pode facilitar a adoção do Scrum e trabalhar com outros *Scrum masters* para aumentar a eficiência do *framework*.

2.1.2 Eventos Scrum

O Scrum define alguns eventos regulares para tratar do andamento do trabalho e evitar a necessidade de encontros não programados. Estes eventos são time-boxed, isso significa que a duração máxima do encontro é definida anteriormente, o que ajuda a evitar o desperdício de tempo com desvios de foco.

O *sprint* é um conceito central do Scrum e consiste em uma iteração, de um mês ou menos, onde um incremento usável do produto é criado. Um novo *sprint* começa imediatamente após a conclusão do *sprint* anterior [Schwaber 2011]. Além do trabalho de desenvolvimento, o *sprint* contém os outros eventos do Scrum: *sprint planning meeting*, daily scrums, *sprint review meeting* e a *sprint retrospective*. Em cada *sprint* é definido o que deve ser feito, um plano para guiar o trabalho e o seu produto resultante; nenhuma mudança que possa afetar estes objetivos, o *sprint goal*, deve ser acolhida. Caso o *sprint goal* se torne obsoleto, em decorrência de decisões da organização ou por questões mercadológicas, é possível cancelar o *sprint*, cuja autoridade de tal ação pertence ao *product owner*. Após o cancelamento, o *product backlog* deve ser reexaminado a fim de escolher os itens que permanecem e reestimá-los. Cancelamentos de *sprint* consomem recursos e costumam ser traumáticos.

No início de cada *sprint*, ocorre o *sprint planning meeting* onde a equipe Scrum é reunida para planejar o *sprint*. O *sprint planning meeting* pode ser dividido em duas partes; na primeira parte será discutido o que será implementado, e na segunda parte a equipe define como implementar os itens escolhidos [Schwaber 2011]. Definir o que implementar será resultado das estimativas das funcionalidades feitas pela equipe de desenvolvimento, do *product backlog* priorizado pelo *product owner*, e da capacidade produtiva da equipe. Ainda nesta primeira parte, o *sprint goal*, o objetivo a ser alcançado após o desenvolvimento dos itens selecionados do *product backlog* para o *sprint*, é determinado. O propósito da segunda parte deste evento é decidir como implementar os itens selecionados na parte anterior. Para isto, a equipe de desenvolvimento decompõe os itens do *backlog* em itens menores. A decomposição facilita o entendimento da equipe, e a presença do *product owner* neste momento evita possíveis equívocos. A saída do *sprint planning meeting* é o *sprint backlog*, que são simplesmente os itens selecionados do *product backlog* para o *sprint* e o plano de como entregá-los.

Daily scrums são reuniões diárias de no máximo 15 minutos onde a equipe de desenvolvimento sincroniza suas atividades e planeja suas ações para o dia seguinte [Schwaber 2011]. As reuniões diárias melhoram a comunicação da equipe e aumentam a probabilidade da equipe de desenvolvimento alcançar o *sprint goal*. Estas reuniões são feitas geralmente no mesmo local e no mesmo horário para reduzir complexidade. Durante a reunião, três questionamentos devem ser abordados por cada membro da equipe:

- O que foi feito desde o último encontro?
- O que será feito até o próximo encontro?
- Existe algum impedimento ao seu trabalho?

O *sprint review* é uma reunião que ocorre no fim do *sprint*. Nesta reunião, a equipe Scrum e os *stakeholders* discutem o que foi feito durante o *sprint*. O *product owner* identifica o que foi feito e o que não foi. A equipe de desenvolvimento demonstra o trabalho realizado no incremento, assim como aborda os fatores de sucesso e de problemas durante o *sprint*. Ainda no *sprint review*, o *product backlog* pode ser revisado e os prováveis itens do próximo *sprint* são definidos [Schwaber 2011].

Outro evento do Scrum é o *sprint retrospective*, uma oportunidade para que a equipe Scrum se avalie e planeje melhorias para o próximo *sprint*. O objetivo é inspecionar o processo, as ferramentas, as pessoas, os relacionamentos; identificar potenciais melhorias; e implementar estas melhorias no próximo *sprint*. É função do *Scrum master* assessorar e encorajar a evolução da equipe Scrum. O *sprint retrospective* ocorre depois do *sprint review* e antes do próximo *sprint planning meeting* e propicia um tempo exclusivo de inspeção e adaptação para a equipe Scrum [Schwaber 2011].

2.1.3 Artefatos Scrum

O Scrum define dois principais artefatos: *product backlog* e *sprint backlog*.

O *product backlog* é uma lista ordenada de todo o trabalho necessário para um projeto [Schwaber 2011]. É um artefato dinâmico, que não reúne somente os requisitos já definidos, mas também os requisitos que surgem ao longo do projeto. A ordenação dos itens considera diversos aspectos, como valor de negócio, risco, prioridade, estimativa e necessidade. O conteúdo e priorização dos requisitos que compõem o *product backlog* são de responsabilidade do *product owner*. A parte de estimativa é de responsabilidade da equipe de

desenvolvimento; durante esta parte é comum que novos detalhes dos requisitos sejam adicionados tanto para tornar as estimativas mais precisas quanto para auxiliar o futuro desenvolvimento. Por ser um artefato dinâmico, o *product backlog* pode se tornar em uma lista maior e mais exaustiva. Mudanças nos requisitos de negócio, condições de mercado, ou tecnologias podem causar mudanças no *product backlog* [Schwaber 2011].

Em qualquer momento do projeto, o *product backlog* informa todo o trabalho necessário para concluir o projeto. Esta informação, adicionada com a informação da velocidade de desenvolvimento da equipe, fornece uma previsão de conclusão do projeto. Esta prospecção pode ser visualizada graficamente através de gráficos de *burndown* e de *burnup*

O topo da lista do *product backlog* reúne os requisitos mais importantes e geralmente os mais detalhados. O conjunto destes requisitos, que caiba dentro de um *sprint*, mais um plano para transformá-los em um incremento formam o *sprint backlog* [Schwaber 2011]. É possível que novos itens sejam adicionados ou que itens já escolhidos sejam excluídos, mas este controle do *sprint backlog* é da equipe de desenvolvimento. De maneira análoga ao *product backlog*, é possível a confecção de gráficos de *burndown* e de *burnup* para fins de monitoramento do andamento do *sprint*.

2.1.4 Ciclo

Para ter uma visão mais ampla de como todos os elementos do Scrum interagem, o ciclo do *framework* é mostrado na Figura 1. Ele se inicia quando o *product owner* define uma lista priorizada com os requisitos. Esta lista é chamada de *product backlog* e é dinâmica, itens podem ser adicionados ou removidos durante o projeto [Scrum Alliance]. Após ter esta lista priorizada, a equipe se reúne para o *sprint planning*, e escolhem os requisitos com prioridades mais altas e discutem como implementá-los. *Sprint backlog* é o nome dado a esta lista dos requisitos escolhidos e que a equipe se compromete a implementar durante a próxima iteração; os itens presentes no *sprint backlog* são mais detalhados pela equipe. A equipe começa então a iteração, que costuma durar entre duas a quatro semanas e é chamada de *sprint* no Scrum. Durante o *sprint* a equipe trabalha para completar os itens do *sprint backlog* e diariamente se reúne para reportar seus progressos e seus planos para o dia seguinte. Estas reuniões diárias são conhecidas como Daily Scrum e devem durar até 15 minutos; para evitar

que o encontro se prolongue, é recomendado que os participantes fiquem em pé. Durante todo o processo o *Scrum master* trabalha para que a equipe esteja focada na conclusão do *sprint*. O objetivo é que no final do *sprint* seja entregue um incremento do software do cliente. O *sprint* termina com o *sprint review* e a *sprint retrospective*, onde é apresentado ao cliente o incremento desenvolvido e a equipe discute os pontos positivos e negativos que ocorreram durante o *sprint*.

Figura 1. Ciclo do Scrum [Scrum Alliance]

Após o término do *sprint*, o ciclo se repete. Um novo *sprint* é iniciado, novos requisitos formam o *sprint backlog* e novas entregas são feitas ao cliente. Este ciclo continua até que todos os itens do *product backlog* estejam completos, ou até o prazo final do projeto.

2.2 Extreme Programming (XP)

Kent Beck foi o criador deste método, que começou a surgir durante um projeto da *Chrysler - Comprehensice Compensation System (C3)* na década de 90. Nos anos seguintes o método foi refinado e em 1999 o livro *Extreme Programming Explained* foi publicado pelo autor do método. Nesta publicação [Beck 1999], o *Extreme Programming (XP)* é definido como um método de desenvolvimento de software focado em projetos conduzidos por equipes pequenas e onde a definição dos requisitos é vaga. Uma segunda edição do livro foi publicada em 2004 [Beck 2004], em que um quinto valor foi adicionado e o conjunto de práticas revisto.

Antes de seguir um conjunto de práticas, o XP define cinco valores que servem para guiar o estilo de desenvolvimento da equipe. Os valores do XP são: comunicação, simplicidade, *feedback*, coragem e

respeito.

Comunicação é o primeiro valor do XP. É importante que haja uma boa comunicação entre todos os envolvidos em um projeto. Quando surge uma dúvida de domínio, por exemplo, a melhor maneira de se resolver isto é o programador discutir abertamente com o cliente e evitar que esta dúvida possa importunar no futuro. Para estimular um bom fluxo de comunicação, o XP define algumas práticas que não podem ser executadas sem comunicação, como a programação em pares [Beck 1999].

Simplicidade é o segundo valor do XP e nem sempre é fácil de ser alcançada. O XP acredita ser melhor fazer algo simples hoje, e absorver uma eventual mudança posteriormente, a fazer algo mais complicado hoje que pode nunca ser utilizado depois [Beck 1999]. Simplicidade e comunicação possuem uma forte relação; quanto maior a comunicação mais fácil fica o entendimento do que realmente precisa ser feito e do que pode ser dispensado; quanto maior a simplicidade do seu sistema menor será a necessidade de comunicação para especificá-lo.

Feedback é o terceiro valor do XP. Um *feedback* constante entre o cliente e a equipe é muito valioso pois permite que erros e desvios sejam identificados tão logo ocorram. Comunicação e simplicidade também funcionam junto com o *feedback*. Se alguém encontra algum problema com o código, basta testar o cenário que irá quebrá-lo, isto é muito mais eficiente do que extensas discussões. Sistemas simples também facilitam a escrita dos testes.

Coragem é o quarto valor do XP. É preciso ter coragem para absorver as grandes mudanças ou assumir os erros quando necessário. A coragem se valoriza quando combinada com os outros três valores. A simplicidade dos sistemas aumenta a coragem na hora de modificar o código, assim como a comunicação e o *feedback* que trazem segurança pois você tem uma noção mais clara do que precisa ser feito.

Respeito é o quinto valor do XP. Esse talvez seja o mais básico de todos os valores e serve de sustentação para os demais. Todos os membros da equipe são importantes e devem ser tratados de forma igual. É preciso compreender e respeitar o ponto de vista do outro para o bom seguimento do projeto.

O XP também apresenta uma lista de práticas que faz com que todos os valores e princípios do método possam funcionar. As práticas não são inéditas ou originais, são na verdade um conjunto onde uma prática fornece suporte à outra. A fraqueza de uma prática é compensada

pela força de outra [Beck 1999]. A segunda versão do XP separa o conjunto de práticas em dois grupos: práticas primárias e práticas corolárias [Beck 2004]. O primeiro grupo reúne as práticas que podem ser implementadas imediatamente para melhorar o desenvolvimento de software. Já o segundo grupo requer um melhor entendimento dos valores do XP e não é seguro que essas práticas sejam implementadas antes das práticas primárias. As práticas primárias serão descritas brevemente abaixo e as práticas corolárias listadas em seguida [Beck 2004].

Sentem-se juntos (Sit together)

Esta prática traz uma idéia muito simples, que é alocar os participantes do projeto em uma mesma sala. A proximidade e o contato visual entre o pessoal do projete favorece a comunicação e aumenta a produtividade.

Equipe inteira (Whole team)

Esta prática consiste em ter uma equipe completa, que tenha todas as habilidades necessárias para o projeto. Se determinada habilidade não está presente, deve-se contratar alguém com esta habilidade. Da mesma forma, se uma habilidade não é mais necessária, a pessoa com essa habilidade deve deixar a equipe.

Ambiente de trabalho informativo (Informative workspace)

A ideia é que o ambiente de trabalho forneça informações sobre o estado do projeto de maneira fácil e intuitiva. Isso pode ser feito através de diagramas ou cartões de histórias de usuários na parede.

Trabalho energizado (Energized work)

Trabalhe apenas o tempo necessário para você ser produtivo e de maneira sustentável. Horas-extra podem trazer benefícios em curto prazo, mas no longo prazo as pessoas se desgastam e perdem produtividade.

Programação em pares (Pair programming)

Toda a codificação ocorre com dois desenvolvedores. Um é responsável por implementar o requisito enquanto o outro acompanha de maneira mais estratégica, pensando no funcionamento e nos testes do código. Esta prática também ajuda quando alguém da equipe tem dificuldades em determinado assunto, pois este pode solicitar a ajuda de alguém mais experiente.

Histórias (Stories)

É uma forma de expressar os requisitos do sistema do ponto de vista do usuário. Para cada funcionalidade do sistema [Beck 1999], o cliente escreve uma história do usuário, com o auxílio do desenvolvedor,

para elucidar informações sobre a funcionalidade do sistema e sua prioridade.

Ciclo semanal (Weekly cycle)

Essa prática recomenda que o planejamento do trabalho seja feito para no máximo uma semana, pois é natural o surgimento de mudanças no sistema e um planejamento mais longo se tornaria desatualizado rapidamente. Além disso, o planejamento semanal ajuda a manter o foco da equipe sobre o que deve ser feito.

Ciclo trimestral (Quaterly cycle)

A ideia é que se façam revisões regulares para discutir questões mais abrangentes, objetivos e prioridades. Também serve para discussões acerca das práticas ágeis utilizadas e novas tecnologias.

Negligência (Slack)

Essa prática sugere a inclusão de algumas tarefas ou histórias do usuário que poderiam não ser implementadas caso a equipe se atrase. Outra forma de implementar essa prática seria através da adição de um tempo livre para que o membros da equipe escolham algumas tarefas por vontade própria.

Construção em dez minutos (Ten-minute build)

Essa prática sugere que os testes e o build do sistema sejam realizados automaticamente no tempo ideal de dez minutos. Builds manuais estão mais suscetíveis a erros e tendem a ocorrer com menos frequência.

Integração contínua (Continuous integration)

O código é integrado e testado continuamente e sempre há uma versão 100% funcional do software para demonstração. Geralmente se usa uma máquina dedicada para esta integração.

Desenvolvimento orientado a testes (Test-first programming)

Os desenvolvedores escrevem testes de unidade para tornar o código mais confiável, o que também aumenta a segurança para absorver novas mudanças. O desenvolvimento é orientado a testes, ou seja, os testes são escritos antes do código do sistema.

Projeto incremental (Incremental design)

Essa prática estimula que a equipe faça o projeto apenas das partes que serão implementadas no ciclo corrente. A ideia é evitar o retrabalho que surgiria para acomodar as mudanças que ocorrem ao longo do tempo.

Práticas corolárias

- Envolvimento real do cliente (Real customer involvement)

- Implantação incremental (Incremental deployment)
- Continuidade da equipe (Team continuity)
- Encolhendo equipes (Shrinking teams)
- Análise da causa raiz (Root-cause analysis)
- Código compartilhado (Shared code)
- Codificação e testes (Code and tests)
- Base de código única (Single code base)
- Implantação diária (Daily deployment)
- Contrato de escopo negociável (Negotiated scope contract)
- Pagamento sob demanda (Pay-per-use)

2.3 Feature Driven Development (FDD)

O Feature Driven Development (FDD) é um método ágil de desenvolvimento criado em 1997 em um grande projeto bancário em Singapura. Nasceu da experiência de Peter Coad e Jeff De Luca e foi inicialmente publicado em 1999. O objetivo do FDD é tornar confiável a entrega contínua de software funcional, de modo que atenda todos os envolvidos, internos e externos, no projeto [De Luca].

Neste método, os requisitos do projeto são chamados de funcionalidades e o desenvolvimento é voltado a elas. Inclusive, é proposto um *template* para a escrita de funcionalidades [De Luca]:

<ação> o <resultado> <por|para|de> um <objeto>

Alguns exemplos:

- Calcular o total a pagar de uma Compra;
- Calcular a quantidade total de vendas de uma Loja para um Produto.

O FDD define duas grandes fases de seu ciclo (Tabela 1): Fase inicial e Fase de construção. Na fase inicial estão compreendidas três atividades: Desenvolver um modelo geral, Construir uma lista de funcionalidades e Planejamento. A fase de construção, por sua vez, é composta por duas atividades que se desenvolvem de maneira iterativa durante o projeto, que são: Projetar por funcionalidades e Construir por funcionalidade [FDD].

Tabela 1. Fases do FDD

Fase inicial	Desenvolver um modelo geral
	Construir uma lista de funcionalidades
	Planejamento
Fase de construção	Projetar por funcionalidade
	Construir por funcionalidade

A Figura 2 ilustra o processo do FDD, com todas as suas atividades. Em seguida, maiores informações sobre cada atividade serão fornecidas.

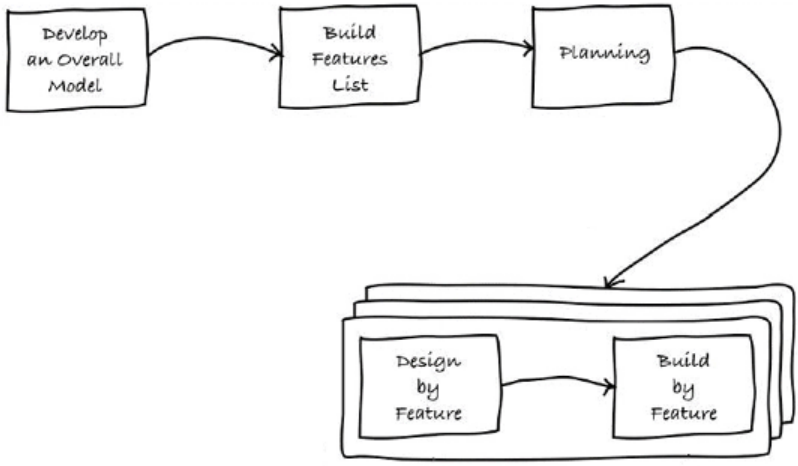


Figura 2. Processo do FDD [De Luca]

2.3.1 Desenvolver um Modelo Geral

Nesta primeira etapa do processo do FDD são discutidas questões sobre o contexto e os requisitos do projeto. O objetivo é produzir uma modelagem geral do domínio da aplicação, podendo inclusive, conter diagramas UML. Esta etapa se distingue da etapa seguinte, pois aqui é obrigatória a presença de especialistas de domínio [De Luca].

2.3.2 Construir uma lista de funcionalidades

Esta atividade objetiva a construção de uma lista completa com todas as funcionalidades do projeto. Uma funcionalidade não deve ser muito ampla e o esforço necessário para a sua construção não deve

ultrapassar duas semanas. Caso essas duas semanas não sejam suficientes, a *feature* deve ser desmembrada em outras. Originalmente, as funcionalidades também eram priorizadas e uma estimativa de tempo de desenvolvimento era realizada, mas atualmente isto não é mais realizado por ser uma atividade muito intensa e que não agrega muito valor [De Luca]. Algumas *features* podem ser subdivididas em listas menores, formando pequenos grupos que compartilham algum tipo de dependência no projeto.

2.3.3 Planejamento

Aqui é definido um plano para a construção dos conjuntos de funcionalidades. Incluem-se também diversos fatores que podem interferir no desenvolvimento das funcionalidades, como riscos do projeto e complexidade.

2.3.4 Projetar por funcionalidade

As iterações da Fase de Construção se iniciam com esta etapa, na qual as funcionalidades são detalhadas para sua posterior implementação, o que pode incluir testes. Ocorre também refinamento do modelo de domínio do projeto e inspeção do projeto.

2.3.5 Construir por funcionalidade

As funcionalidades são implementadas nesta atividade. Testes de unidade e de integração também podem ser desenvolvidos. A inspeção de código também pode ser realizada nesse momento. O resultado desta atividade é um incremento de software que poderá ser integrado ao incremento principal, e possivelmente mostrado ao usuário final.

2.4 Lean Software Development

O *Lean Software Development* (LSD) surgiu inicialmente em um livro, cujo título tem o mesmo nome, escrito por Tom Poppendieck e Mary Poppendieck em 2003 [Poppendieck 2003]. Esse método tem íntima relação com a metodologia *Lean*, originalmente desenvolvida pela Toyota para guiar seus processos de linha de montagem de automóveis. O objetivo do *Lean* pode ser simplificado através da expressão "*Just in time*"; os automóveis passaram a ser montados somente após a concretização de um pedido, eliminando desperdícios com estoque e aumentando a velocidade do processo.

O método LSD é mais voltado para princípios do que para práticas propriamente. O livro sobre LSD [Poppendieck 2003] apresenta

sete princípios que guiam o desenvolvimento de software Lean, assim como 22 ferramentas que ajudam a traduzir esses princípios em práticas ágeis. Comentários breves sobre esses sete princípios são mostrados abaixo.

2.4.1 Elimine o desperdício

O objetivo desse princípio é descobrir tudo o que o cliente deseja e, então, desenvolver e entregar o produto da maneira exata como ele quis, o mais imediato possível. Tudo o que não corroborar para satisfazer a necessidade do cliente é desperdício, como, por exemplo, codificar funcionalidades que não são imediatamente necessárias [Poppendieck 2003].

2.4.2 Aumente o aprendizado

O desenvolvimento de software é um processo de aprendizado contínuo, semelhante ao desenvolvimento de uma receita por um chefe de cozinha. É natural que a receita seja aprimorada ao longo do tempo, assim como o desenvolvimento de software [Poppendieck 2003].

2.4.3 Decida o mais tarde possível

Práticas de desenvolvimento que permitem decisões tardias são mais efetivas em domínios que envolvem incertezas. Melhores decisões podem ser tomadas baseadas em fatos do que em especulações. A capacidade de acolhimento de mudanças em um sistema é importante [Poppendieck 2003].

2.4.4 Entregue o mais rápido possível

Sem rapidez, você não consegue retardar decisões ou obter um *feedback* confiável. Rapidez permite que o cliente obtenha aquilo que ele quer hoje, e não o que queria ontem [Poppendieck 2003].

2.4.5 Dê poder à equipe

Ninguém entende melhor do trabalho do que as pessoas que realmente fazem o trabalho, em um processo de desenvolvimento de software, essas pessoas são os desenvolvedores. Envolver os desenvolvedores nas decisões técnicas é fundamental para alcançar bons resultados [Poppendieck 2003].

2.4.6 Produza integridade

Há dois tipos de integridade: percebida e conceitual [Poppendieck

2003]. Integridade percebida é aquela em que o sistema corresponde aos desejos do cliente; e integridade conceitual significa que os conceitos centrais do sistema são harmônicos e coesos entre si.

2.4.7 Veja o todo

Sistemas complexos requerem conhecimentos em diversas áreas, e é comum que um especialista em uma área tenha a tendência em aperfeiçoar a sua área em vez de aperfeiçoar o sistema como um todo. O pensamento *Lean* precisa ser compreendido por todos para que haja equilíbrio durante todo o desenvolvimento.

2.5 *Framework* de Práticas Ágeis

O *Framework* de Práticas Ágeis foi proposto inicialmente em [Fagundes 2005], e posteriormente estendido em [Machado 2005] com a agregação de práticas ágeis de outros métodos ágeis. O *framework* consiste em um conjunto de práticas ágeis de desenvolvimento de software que permite a escolha das práticas mais convenientes para um determinado projeto no momento da definição do processo [Vilain 2007]. Seu propósito é facilitar o trabalho envolvido na definição de um processo de desenvolvimento ágil próprio, através da seleção de práticas que os envolvidos julguem necessárias. Evita-se, com o uso deste *framework*, a necessidade de avaliar diferentes métodos ágeis e escolher apenas um deles ou um conjunto de práticas deles.

As práticas presentes no *framework* foram analisadas, considerando a fase em que são executadas e suas interdependências, e reunidas a partir dos métodos ágeis mais relevantes: *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD), *Dynamic System Development* (DSDM), *Crystal Clear*, *Lean Software Development* (LSD) e *Agile Modeling* (AM). Uma característica comum dos métodos ágeis é o desenvolvimento iterativo e incremental, e isto influenciou a organização das práticas ágeis no *Framework* de Práticas Ágeis.

No *framework*, o agrupamento das práticas está baseado na lista de atividades do processo de desenvolvimento incremental apresentado em [Sommervile2003]. Estas atividades formam o ciclo de desenvolvimento ilustrado na Figura 3, e acomodam o conjunto de práticas do *framework*. O ciclo pode ser compreendido a partir de três partes. A primeira parte é a fase de iniciação, que ocorre no início do projeto, e contém as atividades “Definição dos Requisitos” e “Projeto da arquitetura do sistema”. A segunda parte é a fase de execução do projeto

e suas atividades são repetidas a cada iteração. Integram esta segunda parte as seguintes atividades: “Atribuição dos Requisitos às Iterações”, “Desenvolvimento do Incremento do Sistema”, “Validação do Incremento” e “Integração do Incremento”. Por fim, tem-se a parte de finalização do ciclo que ocorre após a parte iterativa, onde há as atividades “Validação do Sistema” e “Entrega Final”.

Figura 3. Ciclo do Framework

A seguir, cada uma das práticas presentes no *framework* será descrita em maiores detalhes, seguindo o seu agrupamento por atividades.

2.5.1 Atividade de definição dos requisitos

- Lista de requisitos: Consiste no documento contendo uma lista com os requisitos do sistema. Descrições breves sobre os requisitos e informações adicionais, como prioridade e tempo estimado, podem estar presentes. É importante também a presença do cliente nesta prática. Origem da prática: Scrum e FDD.
- Modelagem geral: Processo de modelagem e documentação dos requisitos do sistema que ajuda no entendimento destes. Origem da prática: FDD.
- Documentação inicial: Alguns documentos são criados para gerir as informações do projeto. Estes documentos

podem incluir a especificação dos requisitos, análise de viabilidade, estimativa de custos, gestão de riscos, entre outros. Origem da prática: AM.

2.5.2 Atividade de projeto da arquitetura do sistema

- Projeto da arquitetura do sistema: Nesta prática é realizado um projeto técnico da arquitetura do sistema de acordo com os seus requisitos. Artefatos como diagramas de classe também podem ser gerados. Origem da prática: XP, Scrum, FDD, DSDM, Crystal Clear, LSD, e AM.

2.5.3 Atividade de atribuição dos requisitos às iterações

- Planejamento da iteração: Aqui ocorre o planejamento da iteração, onde os requisitos que serão desenvolvidos na iteração corrente são escolhidos de acordo com a sua prioridade. Normalmente este planejamento ocorre no início das iterações e na forma de reunião. A equipe de desenvolvimento também deve definir o período de tempo das iterações de acordo com as necessidades do projeto. Para não comprometer os princípios ágeis e satisfazer o cliente com entregas contínuas e frequentes, recomendam-se iterações de uma a oito semanas. Origem da prática: XP, Scrum, FDD, ASD, DSDM, Crystal Clear, and LSD.

2.5.4 Atividade de desenvolvimento do incremento do sistema

- Projeto da iteração: A cada iteração, um projeto do sistema pode ser feito de acordo com os requisitos selecionados na iteração. Artefatos como diagramas de classes e de sequência podem ser gerados. Origem da prática: FDD.
- Histórias do usuário: Para cada funcionalidade do sistema [Beck 1999], o cliente escreve uma história do usuário, com o auxílio do desenvolvedor, para elucidar informações sobre a funcionalidade do sistema e sua prioridade. Um bom resultado aqui alcançado depende da prática Cliente Presente. Origem da prática: XP e Crystal Clear.
- Casos de uso: Esta prática normalmente é uma alternativa à prática de Histórias do usuário. Aqui as

funcionalidades podem ser descritas de forma mais minuciosa através da especificação dos casos de uso e outros diagramas. Esta especificação é feita pelo desenvolvedor, responsável pela geração do artefato, e com o auxílio do cliente, conhecedor das informações de negócio. Origem da prática: FDD.

- Desenvolvimento: Esta prática é considerada obrigatória pelo *Framework* de Práticas Ágeis [Vilain 2007] e compreende a geração de código necessária para atender aos requisitos selecionados na iteração corrente. Para um melhor resultado, deve-se adotar práticas auxiliares como a de Controle de Versões. Origem da prática: XP, Scrum, FDD, ASD, DSDM, Crystal Clear, e LSD.
- Testes de unidade: Testes de unidade são desenvolvidos antes ou durante a codificação do sistema. Se o desenvolvimento orientado a testes, o TDD, estiver sendo seguido, então os programadores devem definir os testes antes da codificação. Origem da prática: XP.
- Testes de aceitação: Os testes de aceitação são escritos a fim de definir objetivamente o que precisa ser conferido para considerar um requisito efetivamente pronto. Nesta prática, é indispensável a participação do cliente. Origem da prática: XP.
- Programação em pares: A codificação do sistema é feita em pares de desenvolvedores. Um deles tem a responsabilidade de desenvolver o código enquanto o outro a de analisar estrategicamente o que está sendo feito; sugere-se a alternância de papéis entre a dupla. Segundo [Beck 1999], a programação em pares diminui a ocorrência de erros no código, o que resulta em melhor qualidade e confiabilidade do código. Origem da prática: XP.
- Desenvolvimento lado a lado: Semelhante à prática de Programação em pares, mas com um computador para cada desenvolvedor e ambos no mesmo local, permitindo que um desenvolvedor veja o trabalho do outro.. Origem da prática: Crystal Clear.
- Refatoração: Prática para melhorar a estrutura interna do código sem alterar o seu comportamento. Sugere-se a refatoração de código sempre que necessário, a fim de

tornar o código mais simples, compreensível e reutilizável. Origem da prática: XP.

- Integração contínua: O código produzido na iteração corrente é integrado simultaneamente ao código do incremento anterior, diminuindo a possibilidade de conflitos e erros no código fonte. Origem da prática: XP e Crystal Clear.
- Desenvolvimento Coletivo do Código: Esta prática define que o conhecimento do sistema e a responsabilidade sobre ele deve ser compartilhada por toda a equipe. Cada participante deve ter conhecimento de todo o sistema, mas isto não implica em saber todos os detalhes do código, implica em ter uma noção do todo e poder trabalhar com qualquer parte do sistema. Origem da prática: XP.
- Reuniões diárias: Nesta prática os integrantes participam de reuniões curtas, em torno de 15 minutos e geralmente em pé, para que todos fiquem informados sobre o andamento do projeto. O objetivo é que cada integrante responda sobre suas tarefas realizadas, suas tarefas futuras e seus possíveis impedimentos. Origem da prática: XP, Scrum e Crystal Clear.

2.5.5 Atividade de validação do incremento

- Inspeção de código: Esta prática tem como objetivo detectar defeitos no código fonte e verificar a sua compreensibilidade. Para se obter uma inspeção imparcial e livre de vícios, é recomendado que um desenvolvedor sempre inspecione o código de outra pessoa e nunca o seu próprio código. Origem da prática: FDD e ASD.

2.5.6 Atividade de integração do incremento

- Reunião de revisão da iteração: Uma reunião é realizada no fim da iteração para que os participantes avaliem o trabalho realizado, assim como apresentem o resultado ao cliente. Origem da prática: Scrum, FDD e LSD.

2.5.7 Atividade de entrega final

- Entrega do sistema: Esta prática só ocorre quando não há

mais requisitos a serem desenvolvidos e/ou o cliente estiver satisfeito com o sistema. Uma reunião com todos os *stakeholders* deve ser realizada para que todos tomem conhecimento da entrega final. Também é considerada uma prática obrigatória. Origem da prática: XP, Scrum, FDD, ASD, DSDM, e Crystal Clear.

- Documentação breve: Caso a equipe tenha optado por não gerar documentação durante o desenvolvimento, é recomendado que se faça uma breve documentação do projeto. Origem da prática: XP, Scrum, AM e Crystal Clear.

3 TRABALHOS RELACIONADOS

A análise de processos ágeis é um assunto ainda não muito explorado na literatura, talvez pelo fato do conceito de agilidade ainda ser muito amplo e permitir diversas abordagens. De qualquer modo, neste capítulo serão apresentados alguns trabalhos relacionados com o tema da proposta deste trabalho.

3.1 An evaluation of the degree of agility in six agile methods and its applicability for method engineering [Qumer 2008]

Um método analítico para avaliar o grau de agilidade de processos de desenvolvimento é apresentado neste trabalho publicado em 2008 [Qumer 2008]. Nele também é constatada a inexistência de uma definição precisa e compreensiva sobre o que é agilidade. Partindo de uma definição contemporânea sobre agilidade, os autores refinam este conceito da seguinte forma:

> *"Um método de desenvolvimento de software é considerado um método ágil quando o método é focado nas pessoas, orientado a comunicações, flexível (pronto para absorver as mudanças esperadas ou inesperadas a qualquer momento), rápido (incentiva o desenvolvimento rápido e iterativo do produto em versões pequenas), leve (focado na redução de prazos e custos e na melhoria da qualidade), sensível (reage apropriadamente às mudanças esperadas e inesperadas), e de aprendizagem contínua (focado na melhoria durante e após o desenvolvimento)"* [Qumer 2008].

A avaliação no nível de agilidade de processos de desenvolvimento de software usa esta definição de agilidade e propõe uma ferramenta denominada 4-DAT. Este *framework* analítico, 4-DAT, avalia os processos sob o prisma de quatro dimensões. A primeira delas é relativa ao escopo do método e abrange os métodos em um mais alto nível, considerando itens como tamanho do projeto, tamanho da equipe, estilo de codificação, entre outros. A segunda dimensão, a única de maneira quantitativa, considera as seguintes características ágeis: flexibilidade, velocidade, leveza, aprendizado e sensibilidade. A terceira e quarta dimensões tratam da caracterização dos valores ágeis e do

processo de software, respectivamente. Abaixo são detalhados quais itens são considerados em cada uma das quatro dimensões.

Tabela 2. Dimensões do 4-DAT

Dimensão 1 (Escopo do Método)
Escopo
1. Tamanho do Projeto 2. Tamanho da Equipe 3. Estilo de Desenvolvimento 4. Estilo de Código 5. Ambiente de Tecnologia 6. Ambiente Físico 7. Cultura Organizacional 8. Mecanismo de Abstração
Dimensão 2 (Caracterização da Agilidade)
Características
1. Flexibilidade 2. Velocidade 3. "Leveza" 4. Aprendizado 5. Capacidade de Resposta
Dimensão 3 (Caracterização dos Valores Ágeis)
Valores
1. Indivíduos e interações mais que processos e ferramentas 2. Software funcionando mais que documentação 3. Colaboração com o cliente mais que negociação de contratos 4. Resposta às mudanças mais que seguir um plano 5. Manter o processo ágil 6. Manter o custo do processo efetivo
Dimensão 4 (Caracterização do Processo de Software)
Processo
1. Processo de desenvolvimento 2. Processo de gerenciamento de projetos 3. Processo de configuração/Processo de suporte 4. Processo de gerenciamento de processos

A maioria destas avaliações é de natureza qualitativa, como, por exemplo, a dimensão 1 que avalia o escopo dos métodos. A Tabela 3 mostra a avaliação do escopo de seis métodos realizada pelos autores.

Tabela 3. Avaliação do escopo de seis métodos ágeis [Qumer 2008]

Criteria	XP	Scrum	FDD	ASD	DSDM	Crystal
Scope						
Project size	Small, medium	Small, medium, and scalable for large	Small, medium, and large (business projects/ applications)	Large and Complex projects	Small and large projects (Business Applications)	Small and medium
Team size	<10	<10 and multiple teams*	No limit – scalable from small to large teams	Not mentioned	Minimum 2 and Maximum 6 (Multiple teams)	Single team in crystal clear with maximum 6 people in a team. Multiple teams with maximum 40 persons in orange and 80 persons in red methodology
Development style	Iterative, rapid	Iterative, rapid	Iterative design and construction	Iterative and rapid development – distributed development	Iterative, rapid development and cooperative	Iterative and rapid development
Code style	Clean and simple	Not specified	Not specified	Not mentioned	Not mentioned	Not mentioned
Technology environment	Quick feedback required	Not specified	Not specified	Not mentioned	Not mentioned	Not mentioned
Physical environment	Co-located teams and distributed teams (limited-interaction)	Not specified	Not specified	Co-located and distributed teams	Not mentioned	Co-located team – no support for distributed development
Business culture	Collaborative and cooperative	Not specified	Not specified	Not specified	Collaborative and cooperative	Not mentioned
Abstraction mechanism	Object-oriented	Object-oriented	Object-oriented	Object-oriented/ Component-oriented	Object-oriented/ Component-oriented	Object-oriented

*This is a special feature.

Somente a segunda dimensão apresenta resultados quantitativos. Nesta dimensão todas as fases e práticas dos métodos citados são valoradas considerando cinco características de agilidade: flexibilidade (FY), velocidade (SD), leveza (LS), aprendizado (LG) e capacidade de resposta (RS). Se a fase ou prática do método atende determinada característica, esta recebe o valor 1, caso contrário recebe o valor 0. Estes valores foram atribuídos de acordo com a literatura. A Tabela 4 mostra a avaliação do XP nesta dimensão.

Tabela 4. Avaliação da agilidade do XP [Qumer 2008]

	Agility features					
XP	FY	SD	LS	LG	RS	Total
Phases						
Exploration	1	1	0	1	1	4
Planning	1	1	0	1	1	4
Iteration to release	1	1	0	1	1	4
Productionizing	1	1	1	1	1	5
Maintenance	1	0	0	1	1	3
Death	0	1	0	0	0	1
Total	5	5	1	5	5	21
Degree of Agility	5/6	5/6	1/6	5/6	5/6	21/(6*5)
Practices						
The planning game	1	1	0	1	1	4
Short release	1	1		1	1	4
Metaphor	0	1	1	0	0	2
Simple design	1	1	1	1	1	5
Testing	1	1	0	1	1	4
Refactoring	1	1	1	1	1	5
Pair programming	1	0	0	1	1	3
Collective ownership	1	0	0	1	1	3
Continuous integration	1	1	1	1	1	5
40-h week	0	0	0	1	0	1
On-site customer	1	0	0	1	1	3
Coding standards	1	1	1	1	1	5
Total	10	8	5	11	10	44
Degree of agility	10/12	8/12	5/12	11/12	10/12	44/(12*5)

Diversos métodos ágeis são avaliados usando esta ferramenta: XP, *Scrum*, FDD, ASD, DSDM e *Crystal*. Dois modelos de processo de software tradicionais, Modelo em cascata e Espiral, também ilustram uma interessante comparação com os métodos ágeis. A Figura 4 mostra a comparação da agilidade, derivada do *framework* 4-DAT, dos seis métodos ágeis e dos dois modelos de processo tradicionais abordados nesse trabalho.

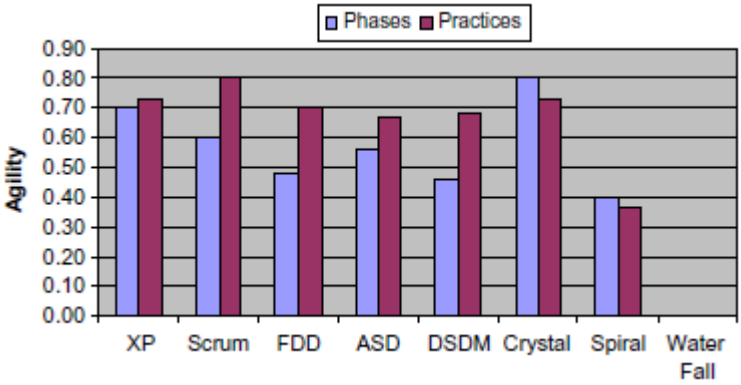


Figura 4. Comparação da agilidade usando o 4-DAT [Qumer 2008]

3.2 Agility measurement index: a metric for the crossroads of software development methodologies [Datta 2006]

Este trabalho foi publicado pela ACM (ACMSE 2006) e propõe a definição de uma métrica para avaliar a agilidade de diferentes metodologias de desenvolvimento de software. O estudo aborda três metodologias de desenvolvimento que surgiram em cenários distintos e com valores e princípios conflitantes. Trata-se do Modelo em cascata, do Processo Unificado e do XP.

O objetivo da definição da métrica, que é denominada *Agility Measurement Index (AMI)*, é orientar a escolha do processo de desenvolvimento que melhor se enquadre em determinado projeto. Para isto, são definidas cinco dimensões para um projeto de desenvolvimento de software: duração, risco, inovação, esforço e interação. **Duração** representa o tempo que um projeto tem até o prazo final. **Risco** aborda o impacto que o software entregue tem no seu cenário de uso; pode ser uma missão crítica como um sistema de monitoramento de um paciente em um hospital ou outro uso relativamente menos sensível. **Inovação** contempla o uso de tecnologias novas pelos desenvolvedores ou o uso de software em domínio que os usuários não estão habituados. **Esforço** é uma estimativa, em homens-hora, do que será gasto durante a execução o projeto. **Interação** diz respeito ao nível de interação – diária, mensal ou só no final do projeto – entre o cliente e a equipe de desenvolvimento.

Para utilizar a métrica, são atribuídos três valores às dimensões supracitadas de acordo com o projeto a ser avaliado (

Tabela *5*). Os três valores correspondem ao valor mínimo (N),

máximo (X) e atual (A) de uma determinada dimensão. O último, A, deve variar em uma escala de N a X. A escolha da escala de N a X é determinada conforme a necessidade de granularidade de uma dimensão. Para cada dimensão, ainda, é calculado o valor de SD, *Specific Dimension*, que corresponde à diferença de A por X.

Tabela 5. Exemplo de cálculo do AMI e SD [Datta2006]

Dimension	N	X	A	SD = A/X
Duration (D)	1	3	1.5	0.5
Risk (R)	1	5	2.5	0.5
Novelty (N)	1	4	1	0.25
Effort (E)	1	6	5	0.83
Interaction (I)	1	10	7	0.7

A partir destes valores, o cálculo do indicador AMI é feito conforme a fórmula abaixo.

$$AMI = \frac{(\sum Actual\ score\ for\ each\ dimension)}{(\sum Maximum\ possible\ score\ for\ each\ dimension)}$$

Valores baixos do indicador indicam que o projeto apresenta curta duração, baixo risco, pouca inovação, esforço limitado e pouca interação com o cliente. Nesse caso, a metodologia mais indicada seria o Modelo em Cascata. Para valores altos é preciso considerar o valor *Specific Dimension (SD)* atribuído a cada dimensão. Projetos com um alto AMI e alto SD para as dimensões Duração e Risco sugerem o Processo Unificado. Em projetos também com alto AMI e alto SD para as dimensões Inovação e Interação o XP seria mais recomendado. No exemplo da

Tabela 5, a metodologia mais indicada seria o XP devido ao alto valor de Interação (0.7). Casos controversos, todavia, podem ocorrer devido à subjetividade na escolha dos valores de cada dimensão.

3.3 Caracterização de Métodos Ágeis de Desenvolvimento de Software [Abrantes 2007]

O objetivo deste trabalho é apresentar uma caracterização básica de métodos ágeis de desenvolvimento de software, reunindo um

conjunto de características que são necessárias para que um método possa ser classificado como ágil através de uma revisão sistemática da literatura.

Durante a revisão, é utilizado protocolo cujo objetivo é identificar as características dos métodos ágeis de maneira geral. As bases de dados eletrônicas do portal do Capes foram utilizadas como fonte, incluindo Compendex EI, IEEEXplore, Inspec, Web of Science e ACM digital library. Após a execução das buscas, foram identificadas as características apresentadas na tabela abaixo.

Tabela 6. Características dos métodos ágeis [Abrantes 2007]

Característica	**Interpretação**
Incrementalidade	Não tentar construir o sistema todo de uma só vez; o sistema é partido em incrementos (pequenas releases com novas funcionalidades) que podem ser desenvolvidos em paralelo em ciclos rápidos; quando o incremento é completado e testado, ele é integrado ao sistema.
Cooperação	Interação aberta e com proximidade entre os vários *stakeholders* (especialmente entre cliente e desenvolvedores); o cliente deve tomar parte ativa no processo de desenvolvimento e prover feedback de forma regular e frequente.
Transparência em clareza	O método é fácil de aprender e modificar e é suficientemente documentado.
Adaptabilidade	Habilidade e capacidade de adaptação rápida do processo para atender e reagir a mudanças de última hora nos requisitos e/ou no ambiente, ou a situações ou riscos não previstos inicialmente.
Iteratividade	Envolve vários ciclos curtos, dirigidos por características do produto, nos quais certo conjunto de atividades é completado em poucas semanas; estes ciclos são repetidos muitas vezes para refinar as entregas.
Auto-organização	As equipes determinam o melhor modo de trabalhar; a equipe tem autonomia para se organizar da melhor forma para completar os itens de trabalho.
Emergência	Os processos, princípios e estruturas de trabalho são reconhecidos durante o projeto ao invés de serem pré-determinados; permite-se que tecnologia e requisitos emerjam ao longo do ciclo de vida do produto.
Períodos de reflexão e introspecção	Reuniões no fim de cada subprojeto ou iteração para os membros da equipe discutirem o que eles estão fazendo bem e o que precisa ser mudado.
Incorporação de feedback rápido	Equipes capazes de procurar e receber continuamente, feedback de modo mais frequente e com mais rapidez (teoria dos sistemas adaptativos complexos).

Modularidade	Característica que permite que um processo seja quebrado em componentes chamados de atividades; modularidade permite que atividades sejam adicionadas ou removidas de um processo quando necessário.
Restrição de prazo	Estabelecimento de limite de tempo para cada iteração programada. Grandes volumes de desenvolvimento são quebrados em múltiplas entregas que possam ser desenvolvidas incremental e concorrentemente de modo previsível.
Parcimônia	Eliminação de perdas ou habilidade de fazer mais com menos recursos; característica que o processo ágil tem, de requerer o mínimo necessário de atividades para mitigar riscos e alcançar metas; deve-se remover todas as atividades desnecessárias no processo de desenvolvimento.
Convergência	Ataque efetivo a todos os riscos que devem ser considerados; como resultado o sistema se torna mais próximo da realidade buscada a cada iteração; à medida que os riscos são atacados pró-ativamente, o sistema está sendo entregue em incrementos.
Orientação a pessoas	Favorecimento de pessoas sobre processos e tecnologias; desenvolvedores são encorajados a aumentar sua produtividade, qualidade e desempenho; a comunicação e a cooperação dentro das equipes de desenvolvimento são consideradas fundamentais e necessárias. As reuniões diárias em pé e os workshops de reflexão dão às pessoas a chance de manifestar suas preocupações.
Colaboratividade	É uma atitude entre membros da equipe de desenvolvimento, entre os quais se encoraja a comunicação para disseminar informação e apoiar integração rápida de incrementos.
Equipes pequenas	O pequeno número de equipes por projeto é necessário para promover o ambiente colaborativo e por requerer menos planejamento para coordenar as atividades dos membros das equipes.
Testes constantes	Para prevenir a degradação da qualidade devido a entregas muito curtas, dá-se alta ênfase a testes do produto ao longo do ciclo de vida. Métodos ágeis requerem testes de integração ao longo do processo de desenvolvimento. Automação dos testes é importante para que as “builds” diárias passem por testes de regressão.
Equipes locais	Para algumas metodologias significa trabalhar na mesma sala ou em salas adjacentes, o que só funciona para equipes de 8 a no máximo 14 pessoas. Todas as metodologias são sensíveis à localização da equipe, pois estão fortemente fundamentadas em canais de comunicação rápidos e ricos, que permitem reduzir a documentação externa a ser construída e mantida.

Cortesia	Utilização de atividades encadeadas para validar e melhorar os produtos de trabalho das atividades anteriores; atividades de cortesia (complimentary) são atividades que trabalham juntas para produzir um resultado melhor do que produziriam individualmente (ex.: escrever user stories, criar testes de aceitação).

Essas características são discutidas no trabalho, sendo algumas consideradas fundamentais e outras subsidiárias. Após a discussão, propõe-se a seguinte caracterização para métodos ágeis [Abrantes 2007]:

> *"Um método para ser caracterizado ágil, deve apresentar, em um grau adequado ao contexto de desenvolvimento de software em que se insere, as características de adaptabilidade, incrementalidade, iteratividade, colaboratividade, cooperação, orientação a pessoas, parcimônia (leanness) e restrição de prazo."*

3.4 The impact of methods and techniques on outcomes from agile software development projects [Parsons 2007]

Este trabalho apresenta uma análise estatística de dados obtidos através de uma pesquisa de opinião online realizada em 2006 por Scott Ambler [Ambler 2006]. A pesquisa continha perguntas a respeito do uso de alguns métodos e práticas ágeis e o efeito deles/delas nos resultados do projeto, em termos de qualidade do software, satisfação do cliente, produtividade da equipe e custo de desenvolvimento. A partir desses dados, os autores exploram novas perspectivas a respeito do impacto desses métodos.

Uma observação inicial dos dados mostra que mais da metade (59,99%) dos respondentes não usa um método ágil, 24,06% usam um método ágil, 11,80% usam dois métodos ágeis e 4,15% usam três ou mais métodos ágeis. O percentual de uso de algumas práticas ágeis, grande parte originária do XP, também são mostrados. Partindo desses dados, os autores se propõem a responder os seguintes questionamentos:

- O uso de métodos ágeis tem efeito positivo nos resultados (custo, produtividade, qualidade e satisfação)?
- Quais são os métodos ágeis mais efetivos?
- Quais são as práticas mais efetivas?

Os usuários da pesquisa eram questionados sobre como a sua

abordagem de desenvolvimento de software afetava os resultados, por meio das quatro variáveis: custo, produtividade, qualidade e satisfação. Para responder a primeira pergunta, as respostas foram separadas entre os usuários que usavam e os que não usavam métodos ágeis. Os resultados dos dois grupos apresentaram diferenças em relação às quatro variáveis, e os autores verificaram a significância dessas variações através do teste estatístico Mann-Whitney U. A conclusão a que chegaram foi que o uso de métodos ágeis tem um efeito positivo na produtividade, qualidade e satisfação, sem apresentar significativa diferença de custos. Percebeu-se também que os resultados dos usuários que usavam mais de um método ágil eram melhores. A partir desta constatação, buscou-se responder a pergunta seguinte.

A intenção da segunda resposta é saber qual par de métodos ágeis apresenta resultados melhores segundo os relatos dos usuários. Foram identificados 24 pares de métodos usados pelos respondentes, mas apenas os pares com um número mínimo de respostas foram considerados. A Figura 5 mostra as combinações de métodos mais comuns, onde a combinação de Extreme Programming e Feature Driven Development foi a mais popular. Após análise semelhante à aplicada na pergunta anterior, verificou-se que não houve variações significativas em termos de custo e satisfação. A combinação de Extreme Programming e Scrum, no entanto, mostrou melhores valores de produtividade e qualidade, o que significa dizer que Extreme Programming e Scrum formam um bom par de métodos para se adotar.

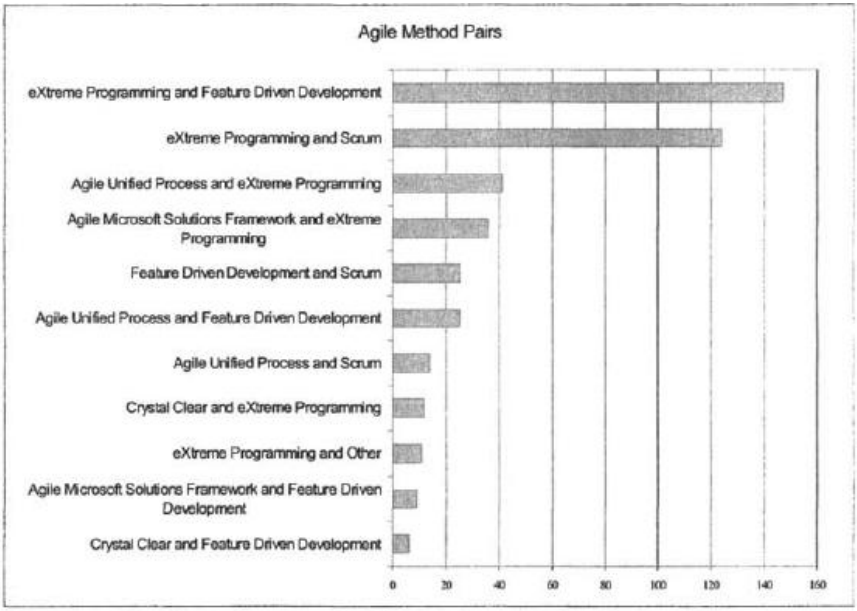


Figura 5. Pares de métodos ágeis mais usados [Parsons 2007]

Um pequeno conjunto de práticas ágeis também forma avaliadas pelos respondentes da pesquisa. Seguindo uma abordagem semelhante à aplicada nas questões anteriores, os autores não identificaram uma variação significativa nas respostas em relação às práticas. Não foi possível, portanto, ter resposta sobre quais práticas são mais efetivas.

3.5 Scott Ambler's March 2007 Agile Adoption Survey [Ambler 2007]

Em 2007, Scott Ambler realizou uma grande pesquisa online sobre a adoção dos métodos ágeis, que foi anunciada no blog do Jon Erickson, o editor do Dr. Dobb's Journal. Esta pesquisa teve bastante representatividade, com 781 respondentes. Deste total de participantes, 52% eram desenvolvedores e 22% tinham funções de gerenciamento; 40% tinham entre 10 e 20 anos de experiência em TI e 33% tinham mais de 20 anos; 33% trabalhavam em organizações com mais de mil pessoas; 85% trabalhavam no setor privado. As perguntas iniciais da pesquisa tratavam de aspectos mais globais do movimento ágil, a fim de saber o seu estágio de desenvolvimento e presença na indústria. Na parte final da pesquisa, havia perguntas sobre a efetividade de diversas

práticas e artefatos originados de diferentes metodologias. Nessas perguntas era possível atribuir um valor acerca da efetividade da prática ou artefato ágil. O valor da efetividade da prática seguia uma escala Likert, de um a cinco pontos, onde tanto maior a efetividade da prática quanto maior o valor da escala. A escala Likert é um tipo de escala psicométrica usada com frequência em pesquisas de opinião, nas quais os respondentes informam o seu nível de concordância com uma afirmação [Likert 1932]. Para os respondentes que preferissem não responder ou que não soubessem avaliar a efetividade da prática, era possível simplesmente não responder ou informar que não sabia. Ao fim da pesquisa, era possível ter uma visão geral da efetividade de diversas práticas ágeis conforme a percepção de um número expressivo de usuários.

As práticas avaliadas pela pesquisa estavam agrupadas em quatro grupos: Práticas de Desenvolvimento, Práticas de Modelagem e Documentação, Práticas de Testes e Qualidade e Práticas de Gerenciamento e Organização. Seguido desses grupos, havia o conjunto dos artefatos. As tabelas a seguir mostram todas as práticas e artefatos (com os nomes preservados em inglês) desses grupos.

Tabela 7. Práticas de Desenvolvimento

Prática
Coding standards
Configuration management
Continuous code integration
Continuous database integration
Data naming conventions
Flexible architectures
Pair programming
Simple design
Static code analysis
Test driven development (TDD)
User interface (UI) refactoring

Tabela 8. Práticas de Modelagem e Documentação

Prática
Architectural spikes
CASE-tool modeling
Evolutionary design
Initial agile architectural modeling
Initial agile requirements modeling
Paper-based modeling (index cards, post its, ...)
Proved the achitecture early in the lifecycle
Whiteboard sketching/modeling

Tabela 9. Práticas de Teste e Qualidade

Prática
Code inspections
Code refactoring
Customer/acceptance tests
Database refactoring
Database testing
Independent confirmatory/exploratory testing
Model/document *review*s
UI testing

Tabela 10. Práticas de Gerenciamento e Organização

Prática
Active stakeholder participation
Collective ownership
Co-located team
Daily stand up *meeting*
Incremental delivery of working software
Iterative development
Planning game
Regular status reports
Self-organizing teams
Small releases
Sustainable pace

Tabela 11. Artefatos

Artefatos
Architecture specification (detailed)
Architecture specification (high-level)
Burn-down chart
Customer/acceptance tests
Defect reports
Defect trend metrics
Developer tests
Gantt chart (detailed)
Gantt chart (high-level)
Iteration task list
Paper models
Requirements specification (detailed)
Requirements specification (high-level)
Source code
Test plan
Use cases (light)
Use cases (detailed)
Velocity
Whiteborad sketches
Working, demoable software

4 PROPOSTA

O *Framework* de Práticas Ágeis [Vilain 2007] permite a definição e personalização de um processo de desenvolvimento de software ágil a partir de um conjunto de práticas originadas de outros métodos. Aspectos sobre a agilidade dessas práticas, contudo, não são consideradas. Surge, então, uma possibilidade a ser explorada: a avaliação da agilidade das práticas do *framework*. Essa avaliação se apresentará em dois momentos. Primeiramente, identificar-se-á a adoção e a efetividade de cada prática ágil presente no *framework* de forma individual a partir da observação dos dados de uma pesquisa online [Ambler 2007]. Posteriormente, e ainda com os dados da mesma pesquisa, serão explorados os relacionamentos e interações entre as práticas na tentativa de identificar grupos de práticas que parecem ser mais efetivos. Pretende-se, com essas avaliações, fornecer indícios que facilitem a escolha das práticas ágeis do *framework* para a definição de um novo processo.

4.1 Observações acerca das práticas ágeis

Como já mencionado, deseja-se identificar a adoção e a efetividade de cada prática ágil do *framework*. Neste trabalho, fica compreendida por **adoção** o quanto determinada prática ágil é usada pela comunidade de desenvolvimento de software, e por **efetividade** o quanto determinada prática tem a capacidade de gerar um efeito positivo no processo de desenvolvimento de software. Optou-se, para esse propósito, utilizar uma abordagem empírica através das informações da pesquisa online [Ambler 2007] realizada por Scott Ambler (seção 3.5).

As observações a respeito das práticas foram possíveis porque o autor da pesquisa disponibilizou os dados brutos de todas as respostas. O universo de práticas utilizado na pesquisa, todavia, é diferente do presente no *framework*, o que torna necessário levantar todas as correspondências das práticas da pesquisa para com as do *framework*. Percebe-se que a pesquisa possui um conjunto de práticas maior que o do *framework*, mas a maioria das práticas do *framework* possui alguma prática correlacionada na pesquisa. Algumas práticas, entretanto, não tiveram uma correspondência direta. Três grupos de práticas foram definidos de acordo com o nível de similaridade da prática do *framework* para com a prática da pesquisa. O primeiro grupo contém as práticas semanticamente idênticas, apenas com pequenas diferenças de nomenclatura. O segundo grupo contém as práticas do *framework* que

não são exatamente iguais às da pesquisa, mas são muito similares; para os fins deste trabalho, as eventuais diferenças entre as práticas da pesquisa e do *framework* foram desconsideradas. Por fim, o último grupo contém as práticas do *framework* que não possuem práticas correlacionadas na pesquisa. É importante registrar que a seleção de artefatos avaliados na pesquisa foi preterida em relação à das práticas, ou seja, quando havia uma prática na pesquisa correspondente a uma prática do *framework*, os artefatos eram desconsiderados por considerar que aquelas apresentam uma abordagem mais abrangente. Foi o que ocorreu com a prática Lista de requisitos, por exemplo, que foi relacionada à prática *Initial agile requirements modeling* em vez do artefato *Requirements specification*. Segue abaixo a relação das práticas do *framework* e suas respectivas práticas na pesquisa.

Tabela 12. Práticas idênticas às da pesquisa

Prática do *framework*	**Prática da pesquisa**
Lista de requisitos	Initial agile requirements modeling
Projeto da arquitetura do sistema	Initial agile architectural modeling
Planejamento da iteração	Planning game
Casos de uso	Use cases (detailed)
Testes de unidade	Test driven development (TDD)
Testes de aceitação	Customer/acceptance tests
Programação em pares	Pair programming
Refatoração	Code refactoring
Integração contínua	Continuous code integration
Reuniões diárias	Daily stand up *meeting*
Desenvolvimento coletivo de código	Collective ownership
Inspeção de código	Code inspections
Desenvolvimento	Obrigatório
Entrega do sistema	Obrigatório

Tabela 13. Práticas muito próximas às da pesquisa

Prática do *framework*	**Prática da pesquisa**
Modelagem geral	Evolutionary design
Histórias do usuário	Use cases (light)
Projeto da iteração	Simple design
Desenvolvimento lado a lado	Co-located team

Tabela 14. Práticas ausentes na pesquisa

Prática do *framework*	**Prática da pesquisa**
Documentação inicial	-
Reunião de revisão da iteração	-
Documentação breve	-

As práticas da Tabela 13, que são as práticas do *framework* que se assemelham às práticas da pesquisa, merecem alguns comentários. No *Framework* de Práticas Ágeis, a prática de Modelagem geral é definida como uma prática a ser realizada na primeira atividade do processo, podendo ser refinada ao longo das atividades seguintes [Vilain 2007]. Por compartilhar esta característica de evolução, esta prática foi associada à de *Evolutionary design*.

Percebe-se também que a prática Histórias do usuário foi relacionada com a prática *Use cases (light)*. Esta relação se explica pelo fato que os casos de uso na versão resumida, não estendida, têm esforço e objetivos semelhantes aos de uma história de usuário. Além disso, existe no *Framework* de Práticas Ágeis, uma prática que compreende a especificação estendida de casos de uso.

Em relação à prática de Projeto de iteração, havia na pesquisa uma série de práticas pertinentes ao projeto. A prática de *Simple design*, oriunda do XP, é a prática em maior conformidade segundo a definição do próprio *Framework* de Práticas Ágeis.

A última prática deste grupo, a de Desenvolvimento lado a lado, foi relacionada à prática *Co-located team* por entender que ambas possuem um propósito semelhante, que é permitir a troca de informações e ajuda mútua entre os integrantes da equipe através da proximidade física entre eles.

O último grupo é composto pelas práticas em que não se identificou nenhuma relação com as práticas da pesquisa.

Antes de prosseguir, é importante ressaltar que estas informações são resultantes de dados obtidos através de pesquisa, em que alguns problemas já conhecidos são inevitáveis. O principal ponto é que os usuários são questionados sobre a efetividade das práticas e respondem muitas vezes baseados em suas opiniões e não exatamente fatos; os critérios adotados na resposta podem ser subjetivos, divergentes de outros usuários e até mesmo contrastantes com a realidade. É natural também que exista uma tendência em parte das respostas em virtude da pesquisa ter sido divulgada em uma comunidade formada por usuários com perfis e pensamentos congruentes com o tema da pesquisa.

4.1.1 Adoção

Algumas observações iniciais a respeito da adoção das práticas ágeis pelos respondentes podem ser feitas. A Figura 6 mostra o percentual de uso de cada uma das práticas. Os valores de um a cinco da escala Likert foram contabilizados como prática em uso; enquanto as respostas não respondidas e em que o respondente não soube responder foram contabilizadas como prática não usada.

Nota-se que a maioria das práticas ágeis teve um percentual de uso entre 40 e 60%, ou seja, quase todas as práticas do *framework* são usadas por cerca de metade dos usuários participantes da pesquisa. As práticas mais usadas, com índices de adoção superiores a 50%, foram: Refatoração, Projeto da iteração, Integração contínua, Desenvolvimento coletivo de código e Testes de aceitação. Nenhuma dessas práticas, porém, se destaca com um índice de adoção muito superior. Três práticas, em situação oposta, tiveram índices ligeiramente menores. Foi o que ocorreu com as práticas: Casos de uso, Planejamento da iteração e Programação em pares. Duas dessas práticas estão presentes no método XP, Planejamento da iteração e Programação em pares. Os baixos índices dessas práticas não se justificaria, portanto, pelo baixo índice de adoção do método usado pelo usuário, visto que o XP é um dos métodos ágeis mais populares. Além disso, outras práticas do XP aparecem com os maiores índices de adoção, como: Desenvolvimento coletivo de código, Refatoração e Integração contínua. O mesmo raciocínio se aplica à prática de Casos de uso. Essa prática está presente no método FDD, que também é composto por uma prática bastante característica como a Inspeção de código, que por sua vez apresentou índice de adoção discretamente inferior a 50%.

Na seção seguinte, observações mais interessantes a respeito da efetividade das práticas serão exploradas.

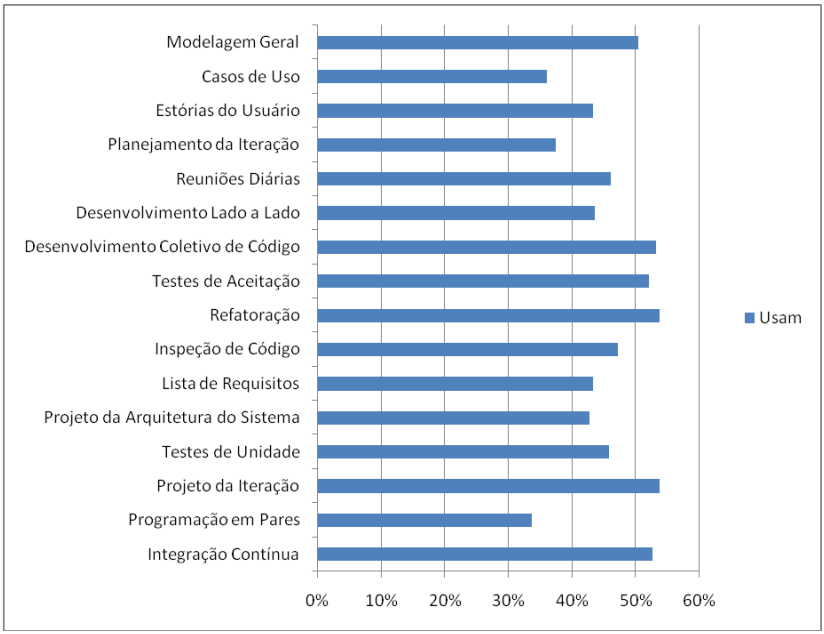


Figura 6. Adoção das práticas ágeis

4.1.2 Efetividade

Os dados da pesquisa [Ambler 2007] mostram na Figura 7 quais práticas do *framework* são mais efetivas e quais são menos efetivas. A tabela abaixo mostra a frequência de cada resposta válida, com valores de um a cinco.

Tabela 15. Frequência das respostas

Resposta	**Frequência**	
1	202	5%
2	369	10%
3	1297	34%
4	965	25%
5	1009	26%
Total	**3842**	**100%**

Percebe-se que a maioria das respostas foi positiva em relação aos efeitos positivos das práticas ágeis e que a resposta mais frequente

foi a de valor médio, o valor três. O valor médio, além de possuir mais de um terço das respostas, pode também ter sido utilizada como uma resposta de dúvida em que o respondente não estivesse convicto sobre a efetividade ou não de determinada prática ágil. Considerando a escala Likert usada, portanto, uma prática foi considerada efetiva quando teve valores quatro ou cinco, enquanto para uma prática ser considerada não efetiva era necessário possuir valores iguais a um ou dois (

Tabela *16*). O valor intermediário da escala Likert foi considerado neutro.

Tabela 16. Práticas efetivas e não efetivas

Escala	**Prática**
5	Efetiva
4	
3	Neutra
2	Não efetiva
1	

A Figura 7 ilustra em termos percentuais a efetividade das práticas do *framework*. Por exemplo, a prática Integração contínua foi considerada efetiva por quase 80% dos usuários, neutra por aproximadamente 15% dos usuários e não efetiva por menos de 10% deles.

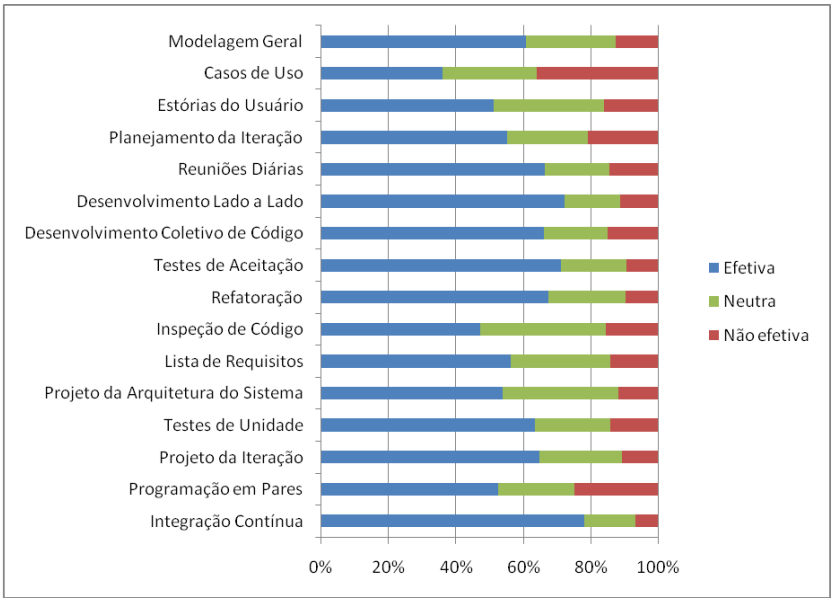


Figura 7. Práticas efetivas e não efetivas

Também é interessante identificar a efetividade em termos absolutos. Para isso, a efetividade das práticas foi calculada com uso dos dados brutos da pesquisa através de uma média com a exclusão dos respondentes que não responderam ou não souberam responder, conforme fórmula apresentada abaixo:

$$E = \frac{(Na \times 5) + (Nb \times 4) + (Nc \times 3) + (Nd \times 2) + (Ne \times 1)}{Tr - Nr - Ns}$$

Onde:

E = efetividade geral da prática;

Na = número de respondentes que atribuíram cinco à efetividade da prática;

Nb = número de respondentes que atribuíram quatro à efetividade da prática;

Nc = número de respondentes que atribuíram três à efetividade da prática;

Nd = número de respondentes que atribuíram dois à efetividade da prática;

Ne = número de respondentes que atribuíram um à efetividade da prática;

Tr = total de respondentes que responderam a pergunta (exclui os que pularam);

Nr = número de respondentes que não responderam a pergunta;

Ns = número de respondentes que não souberam responder a pergunta.

Cada prática do *framework* teve sua efetividade calculada seguindo o mapeamento proposto entre as práticas do *framework* e da pesquisa (

Tabela *12*, Tabela 13 e Tabela 14). A Tabela 17 mostra os valores alcançados. Os valores novamente ficaram muito próximos, mas duas práticas recebem destaque: Integração contínua e Casos de uso. As figuras apresentadas anteriormente já antecipavam e os valores confirmaram: a prática de Integração contínua teve a melhor percepção de efetividade por parte dos respondentes da pesquisa. O outro destaque não é positivo e fica por conta da prática de Casos de uso que, segundo a percepção dos usuários, é a prática menos efetiva do *framework*.

Tabela 17. Efetividade das práticas do *framework*

Prática do FW	**Efetividade**
Lista de requisitos	3,57
Modelagem geral	3,70
Documentação inicial	---
Projeto da arquitetura do sistema	3,60
Planejamento da iteração	3,49
Histórias do usuário	3,52
Casos de uso	2,96
Projeto da iteração	3,78
Testes de unidade	3,79
Testes de aceitação	3,94
Desenvolvimento	---
Programação em pares	3,40
Desenvolvimento lado a lado	4,00
Refatoração	3,81
Integração contínua	4,11
Reuniões diárias	3,88
Desenvolvimento coletivo de código	3,80
Inspeção de código	3,43
Reunião de revisão da iteração	---
Documentação breve	---
Entrega do sistema	---

4.2 Agrupamento das práticas ágeis

Após observar o nível de adoção e efetividade das práticas do *framework*, o interesse passa a ser a exploração das possíveis relações entre as práticas ágeis. Para isso, serão utilizados os dados da mesma pesquisa apresentada anteriormente (seção 3.5) [Ambler 2007]. A partir desses dados, pretende-se investigar quais práticas ágeis, segundo os usuários, costumam funcionar melhor em conjunto, ou seja, possuem maior efetividade. Essa investigação tem como objetivo separar as práticas em grupos que funcionam melhor, ou seja, apresentam maior efetividade, e em grupos que não funcionam tão bem. Com a formação destes grupos será possível visualizar que duas práticas, A e B, quando usadas no mesmo processo, possuem resultados melhores do que o uso de duas outras das práticas, C e D.

Não há, a princípio, nenhuma hipótese sobre o comportamento e

agrupamento dos dados da pesquisa, portanto, com o fim de identificar os grupos de práticas, recorreu-se a uma técnica estatística conhecida como Análise de Agrupamento.

4.2.1 Análise de agrupamentos

A Análise de Agrupamentos é uma técnica estatística que engloba uma variedade de algoritmos cujo objetivo é encontrar e separar objetos em grupos similares [Bassab 1990]. Essa técnica é utilizada em diversas áreas como a medicina, arqueologia e biologia; e o objetivo sempre é classificar um conjunto de dados em grupos que evidenciem suas semelhanças e diferenças.

Existem dois tipos de algoritmos para a classificação dos dados: o método de agrupamento em árvore (tree clustering) e o método de agrupamento por k-médias. A diferença entre esses dois métodos reside no conhecimento prévio a respeito da formação dos grupos. No agrupamento em árvore não há nenhuma hipótese sobre quantos grupos podem ser formados, enquanto que no agrupamento por k-médias você pode informar o número de grupos, os mais distintos possíveis, que deseja formar.

Utilizou-se, nesse trabalho, o método de agrupamento em árvore por não se saber, previamente, o número de grupos a serem formados. Nesse método, os objetos são classificados em grupos em diferentes etapas, de modo hierárquico, produzindo uma árvore de classificação ou dendograma [Bassab 1990]. No dendograma, quanto maior o nível de abstração dos grupos, maior será a diferença entre os elementos de cada grupo. É o que ocorre, de maneira análoga, com a classificação de animais; uma determinada espécie de animal possui características mais próximas em relação a animais da mesma espécie do que em relação a animais do mesmo gênero ou família, onde o grupo abrange um maior número de animais, com maior nível de abstração.

As informações a serem agrupadas são mostradas na Tabela 18. Elas são provenientes da pesquisa online [Ambler 2007] e apresentam a percepção dos dez primeiros respondentes a respeito da efetividade das práticas ágeis do *framework*. Os valores de um a cinco correspondem à efetividade, NA corresponde às perguntas não respondidas e DK às perguntas em que o usuário não soube responder.

Tabela 18. Amostra dos dados brutos da pesquisa

Prática da pesquisa	**r1**	**r2**	**r3**	**r4**	**r5**	**r6**	**r7**	**r8**	**r9**	**r10**
Integração contínua	5	4	NA	1	NA	4	NA	NA	4	4
Programação em pares	4	4	NA	NA	NA	2	NA	NA	NA	5
Projeto da iteração	4	4	NA	1	NA	3	NA	NA	4	5
Testes de unidade	3	1	NA	4	NA	5	NA	NA	NA	4
Projeto da arquitetura do sistema	NA	NA	NA	1	NA	DK	NA	NA	4	3
Lista de requisitos	NA	NA	NA	1	NA	DK	NA	NA	4	3
Inspeção de código	3	2	NA	1	NA	NA	NA	NA	NA	3
Refatoração	5	3	NA	3	NA	3	NA	NA	4	5
Testes de aceitação	4	1	NA	3	NA	3	NA	NA	NA	4
Desenvolvimento coletivo de código	4	4	NA	2	NA	3	NA	NA	2	5
Desenvolvimento lado a lado	5	5	NA	NA	NA	2	NA	NA	NA	4
Reuniões diárias	5	4	NA	1	NA	3	NA	NA	NA	3
Planejamento da iteração	5	4	NA	NA	NA	1	NA	NA	3	3
Histórias do usuário	5	4	NA	3	NA	3	NA	NA	NA	2
Casos de uso	2	4	NA	1	NA	NA	NA	NA	NA	2
Modelagem geral	NA	4	NA	DK	NA	2	NA	NA	4	4

A questão a ser esclarecida é saber quais conjuntos de práticas são mais ou menos efetivos. O uso da Análise de Agrupamentos ajudará a responder essa questão através da unificação de objetos em grupos sucessivamente maiores através de alguma medida de similaridade ou distância. Os grupos, neste caso, são formados pelas práticas, que serão agrupadas de acordo com as respostas dos usuários. O resultado dessa análise será uma árvore hierárquica ou dendograma. A construção da árvore começa através do agrupamento de práticas com as respostas mais semelhantes, para depois iniciar um novo grupo de acordo com a similaridade do grupo formado para com o restante das respostas, até chegar à raiz da árvore, que é um grande grupo com todas as práticas da pesquisa. Por exemplo, se vários usuários consideraram as práticas A e B efetivas, estas duas práticas formarão um pequeno grupo, que se juntará aos demais na medida em que a diferença entre respostas aumentar.

Para construir o dendograma é necessária a utilização de alguma

medida de distância entre os grupos. A distância euclidiana é uma medida bastante utilizada, porém não atende este trabalho, pois ela também agruparia as respostas dos usuários consideradas não efetivas, valores de um a três; se diversos usuários atribuíssem efetividade igual a um para duas práticas, essas duas práticas formariam um grupo. Para evitar a formação de grupos por usuários que considerassem determinada prática não efetiva ou neutra, utilizou-se a medida binária. Essa medida requer que todos os dados a serem analisados estejam em valores binários e apenas os valores iguais a um serão agrupados, descartando os valores iguais a zero, ou seja, mesmo que duas práticas tenham o valor zero de efetividade em diversas respostas, essas práticas não formarão um grupo.

Os dados da Tabela 18 foram transformados para que a medida binária pudesse ser utilizada. Seguindo os parâmetros de efetividade adotados conforme a

Tabela *16*, os dados originais da pesquisa foram transformados em uma nova massa de dados, contendo apenas os valores zero e um. A Tabela 19, por exemplo, é resultado dessa transformação, que teve como entrada a Tabela 18.

Tabela 19. Amostra dos dados tratados da pesquisa

Prática da pesquisa	**r1**	**r2**	**r3**	**r4**	**r5**	**r6**	**r7**	**r8**	**r9**	**r10**
Integração contínua	1	1	0	0	0	1	0	0	1	1
Programação em pares	1	1	0	0	0	0	0	0	0	1
Projeto da iteração	1	1	0	0	0	0	0	0	1	1
Testes de unidade	0	0	0	1	0	1	0	0	0	1
Projeto da arquitetura do sistema	0	0	0	0	0	0	0	0	1	0
Lista de requisitos	0	0	0	0	0	0	0	0	1	0
Inspeção de código	0	0	0	0	0	0	0	0	0	0
Refatoração	1	0	0	0	0	0	0	0	1	1
Testes de aceitação	1	0	0	0	0	0	0	0	0	1
Desenvolvimento coletivo de código	1	1	0	0	0	0	0	0	0	1
Desenvolvimento lado a lado	1	1	0	0	0	0	0	0	0	1
Reuniões diárias	1	1	0	0	0	0	0	0	0	0
Planejamento da iteração	1	1	0	0	0	0	0	0	0	0
Histórias do usuário	1	1	0	0	0	0	0	0	0	0
Casos de uso	0	1	0	0	0	0	0	0	0	0
Modelagem geral	0	1	0	0	0	0	0	0	1	1

Mesmo que tenham sido consideradas as relações da

Tabela *16*, diferentes dendogramas foram explorados através de diferentes configurações – a

Tabela *16* corresponde à configuração B -, cada uma delas com diferentes agrupamentos das respostas, como pode ser visto na tabela abaixo. As respostas efetivas foram mapeadas para o valor um e as respostas não efetivas e neutras foram mapeadas para o valor zero.

Tabela 20. Configurações de agrupamento das respostas

	Configuração A	**Configuração B**	**Configuração C**
Respostas	**Práticas**		
5	Efetiva	Efetiva	Efetiva
4			Neutra
3		Neutra	
2	Não efetiva	Não efetiva	Não efetiva
1			

Para realizar a análise de agrupamento hierárquico dos dados da pesquisa, foi utilizado o programa de estatística computacional R [R Project]. A medida binária foi utilizada conforme justificativa já apresentada, e o dendograma gerado é mostrado na seção seguinte.

4.2.2 Análise do dendograma

A partir dos algoritmos escolhidos, chegou-se aos três dendogramas apresentados nas Figura 8, 9 e 10. Esses são compostos pelas práticas ágeis do *framework*, que estão representadas pelos números de 1 a 16, conforme a correspondência número-prática da Tabela 21.

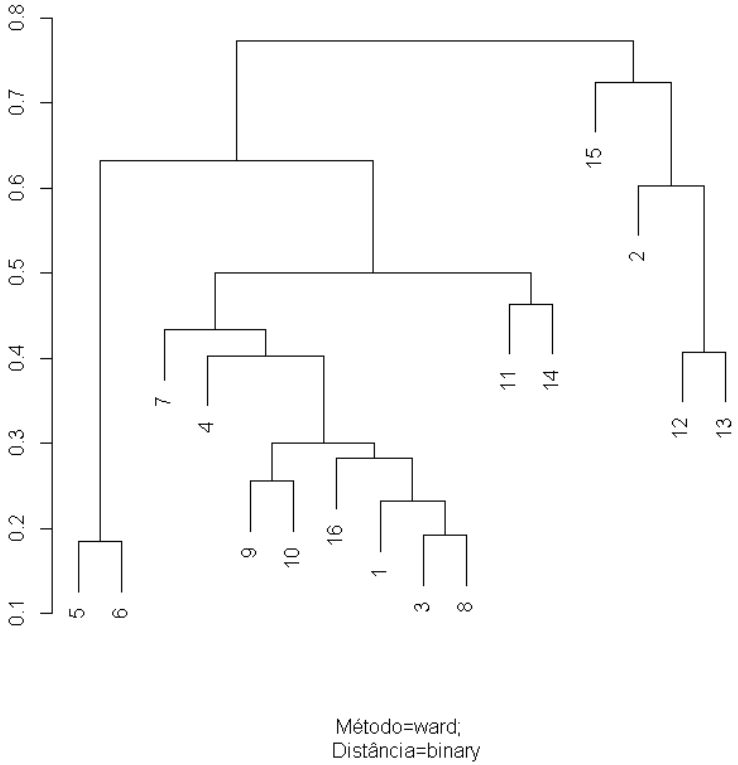


Figura 8. Dendograma das práticas na configuração A

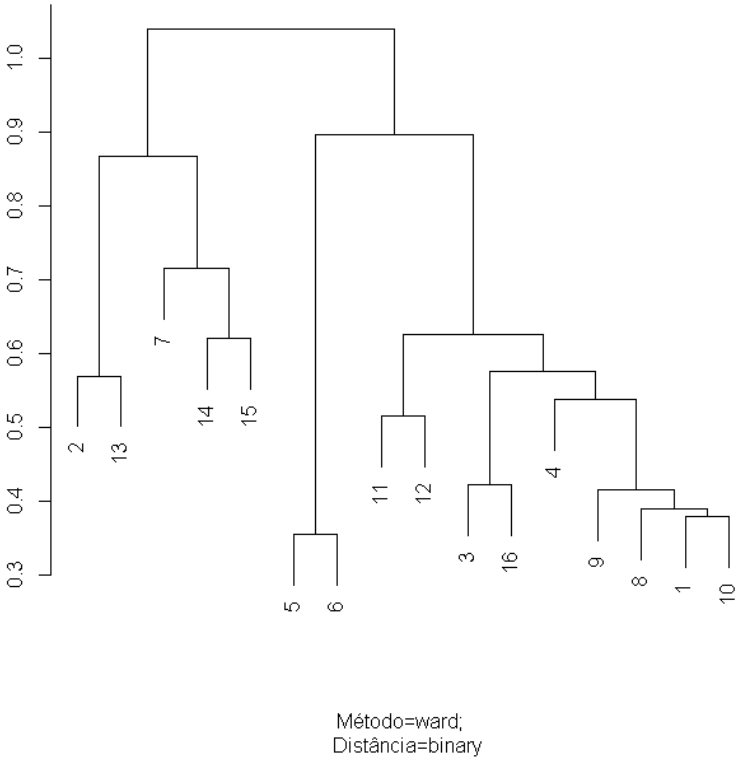


Figura 9, Dendograma das práticas na configuração B

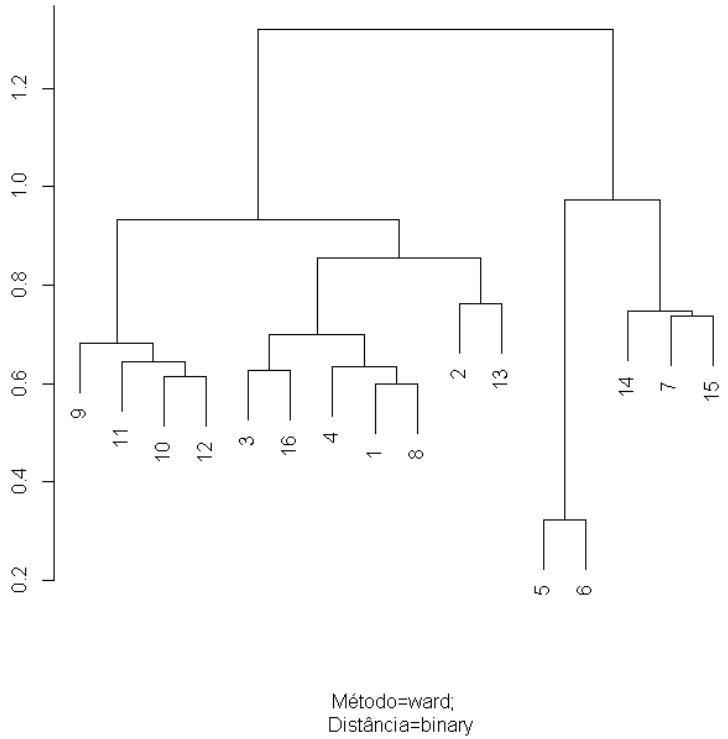


Figura 10. Dendograma das práticas na configuração C

Tabela 21. Práticas do dendograma

#	**Prática**
1	Integração contínua
2	Programação em pares
3	Projeto da iteração
4	Testes de unidade
5	Projeto da arquitetura do sistema
6	Lista de requisitos
7	Inspeção de código
8	Refatoração
9	Testes de aceitação
10	Desenvolvimento coletivo de código
11	Desenvolvimento lado a lado
12	Reuniões diárias
13	Planejamento da iteração
14	Histórias do usuário
15	Casos de uso
16	Modelagem geral

Na medida em que move para a raiz da árvore, agrupamentos cada vez maiores são formados, com cada vez maior diversidade nas respostas dos usuários, ou seja, diminui o consenso acerca da efetividade daquele conjunto de práticas. Os grupos com menor altura (eixo y – Height), portanto, formam os grupos com as práticas que os usuários consideraram mais ágeis. Na formação desses grupos é preciso utilizar um ponto de corte, que tem como propósito separar os itens do dendogrma de acordo com a sua similaridade.

É possível notar em todos os dendogramas que não houve um agrupamento que representasse a efetividade de um método ágil por completo. Práticas do XP (programação em pares e integração contínua) tiveram diferentes percepções de efetividade, por exemplo. É possível concluir que, na pesquisa, as pessoas escolhem as práticas que elas acham mais importantes, independentes se estão utilizando um método por completo ou não.

Os resultados dos dendogramas mostram que a configuração B foi uma boa escolha pelos seguintes motivos: a configuração C apresentou práticas muito próximas, o que formaria um grande grupo com diversas práticas; e a configuração A apresentou um grupo menor que a configuração C, porém ainda grande, de práticas e o ponto de

corte resultaria em apenas um grande grupo.

De acordo com a altura (eixo y), temos um grupo mais uniforme ou efetivo. Considerando a configuração B, por exemplo, o grupo formado pelas práticas 9, 8, 1, e 10 apresentou respostas mais uniformes a respeito da efetividade dessas práticas se comparado com o grupo formado pelas práticas 11 e 12. O dendograma também mostra que o grupo (5+6) apresentou o maior nível de agrupamento (height = 0.35), o que significa dizer que as práticas desse grupo costumam ser usadas em um mesmo processo e são consideradas efetivas. Aumentando o nível de abstração (height = 0.90), considerando menos uniformidade nas respostas, esse mesmo grupo se juntaria a outro grande grupo formado pelas práticas 11, 12, 4, 3, 16, 9, 8, 1, 10. A definição do ponto de corte (height) do dendograma definirá o número de grupos de práticas formados. Quanto menor o ponto de corte, maior será o consenso das respostas sobre a efetividade daquele grupo. Um ponto de corte por volta de 0.50 (Figura 11) formaria os quatro grupos da Tabela 22.

Tabela 22. Grupos de práticas formados pelo dendograma

Grupo	Id.	Práticas
A	5+6	Projeto da arquitetura do sistema
		Lista de requisitos
B	10+1+8+9	Desenvolvimento coletivo de código
		Integração contínua
		Refatoração
		Testes de aceitação
C	3+16	Projeto da iteração
		Modelagem geral
D	11+12	Desenvolvimento lado a lado
		Reuniões diárias

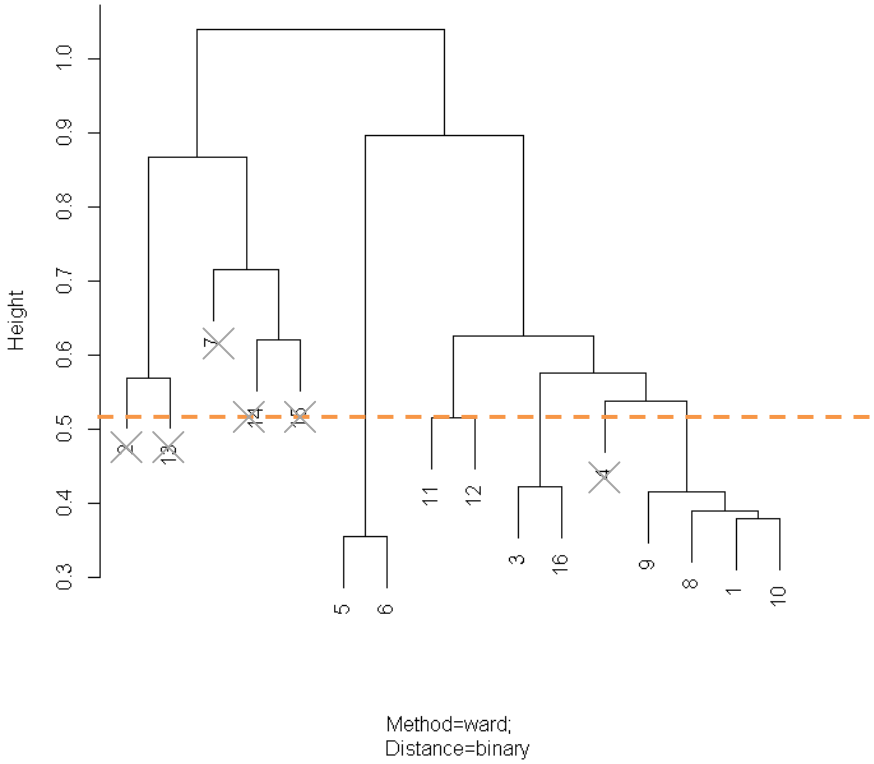


Figura 11. Dendograma e o ponto de corte

É importante buscar maiores indícios que confirmem a relevância desses grupos e a posição do ponto de corte. Para atingir esse propósito, planejou-se a execução de um experimento que abordasse a utilização das práticas ágeis, além de uma pesquisa bibliográfica que pudesse corroborar com as informações apresentadas. Devido à impossibilidade em se testar todas as práticas, foram selecionadas três práticas de modo que duas estivessem em um mesmo grupo, enquanto que a prática restante não estivesse. Esse experimento e os seus resultados são descritos no capítulo 5.

4.3 Seleção das práticas ágeis

As informações obtidas nas seções anteriores foram resumidas na Tabela 23, com exceção a respeito dos grupos de práticas que serão posteriormente adicionados. A primeira coluna da Tabela *23* corresponde ao número da prática ágil; a segunda coluna mostra o nome completo da prática ágil; as colunas seguintes apresentam a adoção e as efetividades da prática respectivamente. A coluna de efetividade corresponde ao percentual/número de respondentes que utilizam determinada prática e a consideram efetiva. A prática lista de requisitos, por exemplo, é adotada por 43% dos respondentes e desses 43%, 56% a consideram efetiva.

Tabela 23. Quadro resumo das práticas

#	Prática do FW	Adoção	Efetividade %[1]	Efetividade[2]
1	Lista de requisitos	43%	56%	3,57
2	Modelagem geral	50%	61%	3,70
3	Documentação inicial	---	---	---
4	Projeto da arquitetura do sistema	43%	54%	3,60
5	Planejamento da iteração	37%	55%	3,49
6	Histórias do usuário	43%	51%	3,52
7	Casos de uso	36%	36%	2,96
8	Projeto da iteração	54%	65%	3,78
9	Testes de unidade	46%	63%	3,79
10	Testes de aceitação	52%	71%	3,94
11	Desenvolvimento	---	---	---
12	Programação em pares	37%	52%	3,40
13	Desenvolvimento lado a lado	43%	72%	4,00
14	Refatoração	54%	67%	3,81
15	Integração contínua	53%	78%	4,11
16	Reuniões diárias	46%	67%	3,88
17	Desenvolvimento coletivo de código	53%	66%	3,80
18	Inspeção de código	47%	47%	3,43
19	Reunião de revisão da iteração	---		---
20	Documentação breve	---	---	---
21	Entrega do sistema	---	---	---

Efetividade %[1] = percentual dos usuários que consideram a prática ágil, conforme Figura 7;

Efetividade[2] = média da efetividade da prática, conforme Tabela 17.

As informações mais relevantes da

Tabela *23* são as colunas sobre a efetividade. Essas informações servirão para definir uma escala de cores para as práticas do *framework* de acordo com a efetividade percebida pelos usuários. A prática menos efetiva será representada pela cor vermelha e a mais efetiva pela cor verde. As práticas intermediárias receberão uma gradação dessas duas cores conforme a efetividade. O objetivo do uso das cores é facilitar a escolha das práticas no momento da definição do processo de desenvolvimento de software.

A **Erro! Fonte de referência não encontrada.** também mostra

os agrupamentos de práticas através de um cruzamento em forma de matriz. As práticas de um mesmo grupo são identificadas através das letras A, B, C, e D. A prática Lista de requisitos, por exemplo, forma um grupo com a prática Projeto da arquitetura do sistema.

Tabela 24. Quadro para seleção de práticas

	Lista de requisitos	Modelagem geral	Documentação inicial	Projeto da arquitetura do sistema	Planejamento da iteração	Histórias do usuário	Casos de uso	Projeto da iteração	Testes de unidade	Testes de aceitação	***Desenvolvimento***	Programação em pares	Desenvolvimento lado a lado	Refatoração	Integração contínua	Reuniões diárias	Desenvolvimento coletivo de código	Inspeção de código	Reunião de Revisão da Iteração	Documentação Breve	***Entrega do Sistema***
Lista de requisitos				A																	
Modelagem geral								C													
Documentação inicial																					
Projeto da arquitetura do sistema	A																				
Planejamento da iteração																					
Histórias do usuário																					
Casos de uso																					
Projeto da iteração		C																			
Testes de unidade																					

Testes de aceitação														B	B		B				
Desenvolvimento																					
Programação em pares																					
Desenvolvimento lado a lado																D					
Refatoração										B					B		B				
Integração contínua										B				B			B				
Reuniões diárias													D								
Desenvolvimento coletivo de código										B				B	B						
Inspeção de código																					
Reunião de Revisão da Iteração																					
Documentação Breve																					
Entrega do Sistema																					

Legenda:

Prática mais efetiva				Prática menos efetiva

A tabela acima visa auxiliar a definição do novo processo. O responsável por essa definição, a partir do *Framework* de Práticas Ágeis, deverá seguir alguns passos ao usar a **Erro! Fonte de referência não encontrada.**:

1. Selecionar as práticas obrigatórias do *framework*: Desenvolvimento e Entrega do sistema;
2. Escolher as práticas desejadas, com o auxílio das cores que indicam a efetividade das práticas individualmente. As práticas verdes, mais efetivas, devem ser beneficiadas e as práticas vermelhas, menos efetivas, preteridas;
3. Considerar algumas diretrizes definidas para auxiliar a definição do processo;
    a. Se o responsável não selecionou todas as práticas do grupo B (Integração contínua, Desenvolvimento lado a lado, Refatoração, Testes de aceitação e Desenvolvimento coletivo de código), sugere-se que elas sejam selecionadas. Esta diretriz é baseada no fato de que grupo contém as práticas com mais alto nível de efetividade (Integração contínua - 78%, Testes de aceitação - 71%, Refatoração - 67%, e Desenvolvimento coletivo de código - 66%), e também por estarem agrupadas no dendograma.
    b. Se o responsável selecionou alguma das práticas do grupo A (Lista de requisitos e Projeto da arquitetura do sistema), mas não todas as práticas do grupo, sugere-se a seleção das outras práticas do grupo. Esta diretriz considera a análise de agrupamento executada na seção anterior.
    c. Se o responsável selecionou alguma das práticas do grupo C (Modelagem geral e Projeto da iteração), mas não todas as práticas do grupo, sugere-se a seleção das outras práticas do grupo. Esta diretriz também considera a análise de agrupamento.
    d. Se o responsável selecionou alguma das práticas do grupo D (Desenvolvimento lado a lado e Reuniões diárias), mas não todas as práticas do grupo, sugere-se a seleção das outras práticas do

grupo. Esta diretriz também considera a análise de agrupamento.

A ideia não é estabelecer conjuntos pré-definidos de práticas a serem utilizados, o que diminuiria a flexibilidade proposta originalmente pelo *framework*. Pretende-se simplesmente auxiliar o processo de escolha das práticas com informações mais empíricas sobre elas e, adicionalmente, sugerir outras práticas consideradas uma boa combinação pelos usuários da pesquisa.

É importante observar também que a **Erro! Fonte de referência não encontrada.** não considera diversos aspectos que influenciam a definição de um processo de desenvolvimento, como, por exemplo, os requisitos e natureza do projeto, as políticas organizacionais e a cultura da equipe e organização.

5 AVALIAÇÃO

O dendograma (Figura 9) apresentado no capítulo anterior ilustra o agrupamento de práticas ágeis de acordo com o seu nível de agrupamento. Este nível de agrupamento é diretamente proporcional ao número de usuários que usaram estas práticas em conjunto e a consideraram efetivas quanto a sua agilidade. O uso de práticas ágeis agrupadas por um maior nível de agrupamento, por dedução, torna o desenvolvimento mais efetivo; esta é a questão a ser trabalhada aqui.

A fim de confirmar a relevância dos agrupamentos do diagrama, foi realizado um experimento com a utilização de algumas práticas ágeis presentes no diagrama (Figura 12). Por questões de disponibilidade de recursos e de logística foram selecionadas apenas três práticas, que formam dois pares. O primeiro par, Refatoração e Testes de aceitação, estava presente no grupo com o maior nível de agrupamento do diagrama junto com as práticas de Integração contínua e de Propriedade Coletiva de Código. O outro par inclui novamente a prática de Refatoração e inclui a prática de Programação em pares; estas foram selecionadas para contrastar com o nível de agrupamento das duas primeiras práticas; elas estavam mais distantes no gráfico e faziam parte de um grupo com maior número de práticas, com menor nível de agrupamento. Tem-se, então, dois pares de práticas com diferentes níveis de agrupamento. Os detalhes sobre a execução do experimento podem ser vistas no (APÊNDICE A).

O experimento, porém, apresentou alguns problemas que influenciaram os resultados. Por esse motivo, o experimento foi posteriormente descartado. Um significativo problema foi a condução da prática de refatoração. A princípio, a refatoração consiste em alterar a estrutura interna do código sem alterar o seu comportamento externo e para isso se faz necessário a existência de testes de unidade, que servem para guiar e validar a correta refatoração. Durante o experimento, não foram feitos testes de unidade e diagramas UML foram entregues aos envolvidos para guiar e facilitar o trabalho dos alunos, o que não está conformidade com a definição de refatoração [Fowler 1999]. Além desse problema, a inexperiência dos alunos e seus conhecimentos não homogêneos refletiram em grande variabilidade dos resultados.

Após a falha com o experimento, percebeu-se a importância da escolha das práticas e a melhor forma de avaliá-las. Muitas práticas se relacionam entre si, como a refatoração e os testes de unidade, e essas relações precisam ser consideradas. A complexidade de cada prática

também é variável; a prática de testes de unidade, em geral, exige algum tipo de treinamento e prática para que dificuldades técnicas não comprometam os resultados. É crucial também que se busque o maior grau de homogeneidade possível entre os participantes do experimento, caso contrário, diferentes graus de compreensão sobre os conhecimentos exigidos podem mascarar a real razão dos resultados. Enfim, o ideal é que se busque um ambiente o mais neutro possível para que esse não influencie os resultados, que devem ser exclusivamente obtidos através do uso das práticas ágeis.

Como forma de avaliação da proposta, então, realizou-se uma revisão sistemática da literatura para confirmar os resultados observados na pesquisa. A revisão abordou apenas as práticas do experimento (Refatoração, Testes de aceitação e Programação em pares).

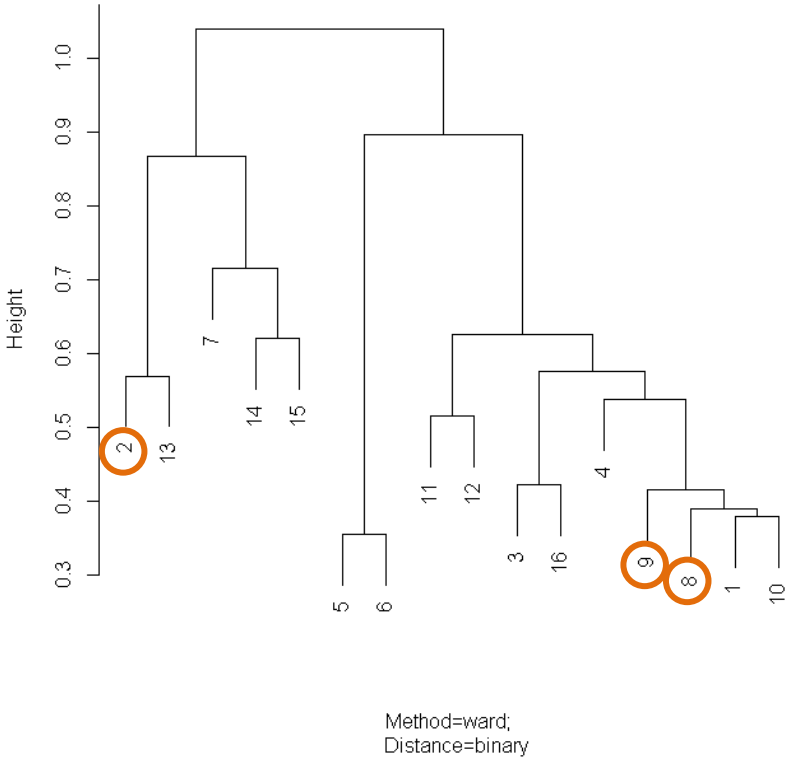


Figura 12. Dendograma e as práticas do experimento

5.2 Revisão sistemática da literatura

É oportuna a realização de uma revisão sistemática da literatura a respeito da efetividade das três práticas abordadas no experimento: refatoração, testes de aceitação e programação em pares. Embora o mais adequado fosse a realização de uma revisão sistemática sobre a efetividade de todas as práticas, esse trabalho se limita em revisar as três práticas.

Pretende-se com a revisão sistemática sumarizar o conhecimento existente sobre a efetividade das práticas ágeis. O processo utilizado para a revisão é apresentado a seguir, que teve como orientação as diretrizes apresentadas em [Kitchenham 2007].

5.2.1 Protocolo da revisão

O objetivo da revisão é identificar a efetividade do uso das práticas refatoração e testes de aceitação ou refatoração e programação em pares, quando usadas em um mesmo processo. A pergunta a ser respondida é: qual a efetividade, positiva ou negativa, em se usar as práticas refatoração e testes de aceitação ou as práticas refatoração e programação em pares em um mesmo processo de desenvolvimento de software? O problema é encontrar informações a respeito da efetividade das três práticas. A aplicação é a comparação dos resultados da revisão com os resultados obtidos através da análise de agrupamentos.

Considerou-se como população o desenvolvimento ágil de software. A intervenção são as práticas refatoração e testes de aceitação e a comparação são as práticas refatoração e programação em pares. O resultado esperado é encontrar a efetividade dos dois pares de práticas.

As fontes de dados serão os arquivos indexados por duas bases de dados eletrônicas: IEEEXplore e ACM *digital library*. O idioma inglês foi escolhido devido à sua expressiva quantidade de trabalhos publicados e indexados.

Foram escolhidas as seguintes palavras-chave:

- População: agile, software, development;
- Intervenção: refactoring, acceptance tests, functional tests;
- Comparação: refactoring, pair programming;
- Resultado: effectiveness.

Utilizaram-se, como critérios de inclusão e exclusão, os trabalhos disponíveis online e que contemplem análises acerca da efetividade das práticas. Como estratégia de extração de informações: título, ano de publicação, fonte e considerações acerca da efetividade das práticas.

O texto de busca utilizado nos sistemas das bases de dados foi:

(agile OR software OR development) AND
(((refactoring AND "acceptance test") OR
(refactoring AND "functional test")) OR
(refactoring AND "pair programming"))
AND (effectiveness)

5.2.2 Execução das buscas

O texto de busca apresentado anteriormente foi utilizado nos sistemas de busca das duas bases de dados eletrônicas. Foram retornados 174 resultados (ANEXO III), 143 da IEEEXplore e 31 da ACM digital

library. Alguns poucos resultados foram excluídos por estarem duplicados ou inacessíveis.

Em um primeiro momento, todos os trabalhos foram avaliados brevemente através da leitura do seu título e *abstract*. A avaliação consistiu em priorizar o texto, de acordo com a pertinência do assunto que se procurava, em uma escala de zero a três. A prioridade zero foi atribuída aos trabalhos replicados, com acesso restrito e que não tratavam do tema. Textos que abordavam o assunto de desenvolvimento ágil de forma superficial ou apenas no nível do método, não no nível de práticas, foram priorizados com a prioridade um. A prioridade dois foi usada para os trabalhos que incluíam análises no nível de práticas ágeis, mas não as pretendidas com essa busca: refatoração, testes de aceitação e programação em pares. Os trabalhos que abordavam a efetividade dessas três práticas receberam a prioridade mais alta: três.

Os trabalhos com prioridade zero foram sumariamente eliminados; 27 trabalhos foram descartados. Uma avaliação mais cuidadosa, que incluiu a leitura do resumo e da parte introdutória, foi realizada a fim de corrigir algum eventual equívoco na priorização preliminar. Após essa avaliação, um total de 24 trabalhos recebeu prioridade três. Esses trabalhos foram analisados por completo.

Dos 24 documentos com prioridade três, apenas oito foram considerados úteis, por discutirem a efetividade das práticas de interesse, e incluídos nesse trabalho: [Haugset 2008], [Khramov 2006], [Melnick 2004], [Muller 2001], [Sfetsos 2010], [Sison 2007], [Usha 2009], [Vanhanen 2005]. Nenhum deles, todavia, fornecia uma resposta para a pergunta formulada nessa revisão. A efetividade da interação entre os pares de práticas (refatoração com testes de aceitação e refatoração com programação em pares) não foi abordada diretamente por nenhum dos trabalhos. Alguns deles incluíam a avaliação de várias práticas, incluindo as de interesse da revisão. Outros trabalhos também abordavam a especificidade de uma única prática, como programação em pares. Mesmo que o foco não seja a efetividade de uma prática individualmente, esses trabalhos foram considerados devido à baixa quantidade de documentos encontrados.

5.2.3 Análise dos resultados

Os resultados encontrados nos oito documentos estão compilados nas tabelas abaixo.

Documento	[Haugset 2008]
Tipo de estudo	Estudo de caso e revisão da literatura.
Prática	Testes de aceitação.
Resultados	Melhora o entendimento do domínio do sistema em desenvolvimento.
	Ajuda a identificar e corrigir mais erros.

Documento	[Khramov 2006]
Tipo de estudo	Estudo de caso.
Prática	Refatoração.
Resultados	A prática não deve ser mais usada quando o sistema está completamente funcional em um nível aceitável de eficiência.
Prática	Programação em pares.
Resultados	Não deve ser usada para tarefas triviais.
	Funciona bem em equipes homogêneas e em projetos com alguns desafios esporádicos.
	É a prática mais controversa do XP.

Documento	[Melnick 2004]
Tipo de estudo	Pesquisa.
Prática	Programação em pares.
Resultados	Considerada útil.
	Apresenta algumas dificuldades de ajuste quando há uma grande diferença de habilidades entre os desenvolvedores.

Documento	[Muller 2001]
Tipo de estudo	Estudo de caso.
Prática	Programação em pares.
Resultados	Apesar da fácil adoção, não se sabe em que tipo de trabalho é adequado.

Documento	[Sfetsos 2010]
Tipo de estudo	Revisão da literatura.
Prática	Programação em pares.
Resultados	Os resultados sobre produtividade e tempo gasto foram contraditórios.
	A prática foi considerada uma boa prática ágil.
	Em conjunto com as práticas TDD e refatoração, torna-se uma prática chave para a melhora da qualidade.
	Reportou-se aumento de esforço e custo.
Prática	Refatoração
Resultados	A prática favorece o aumento da qualidade.

Documento	[Sison 2007]
Tipo de estudo	Estudo de caso.
Prática	Programação em pares.
Resultados	O principal benefício foi acelerar o aprendizado dos programadores iniciantes.
	A necessidade da prática diminuiu à medida em que os programadores ganhavam experiência.

Documento	[Usha 2009]
Tipo de estudo	Experimento.
Prática	Refatoração.
Resultados	Melhora a reusabilidade e legibilidade do código e sua consequente manutenção.

Documento	[Vanhanen 2005]
Tipo de estudo	Experimento.
Prática	Programação em pares.
Resultados	Apresentou produtividade 29% inferior, possivelmente devido à curva de aprendizado.
	Mais recomendada para tarefas complexas.
	Apresenta códigos com menos erros.

As considerações a respeito da prática de refatoração foram boas, sendo duas positivas e uma neutra. Constatou-se que o uso da prática provê melhora a qualidade do sistema e do código.

Apenas um trabalho abordou a prática de testes de aceitação e apresentou duas considerações positivas, sendo que uma delas estava relacionada à diminuição do número de erros. Os resultados dos experimentos mostraram conclusões congruentes, com um menor número de falhas nos testes executados nos grupos que usaram essa prática.

A prática de programação em pares apresentou considerações controversas: cinco considerações positivas, três neutras e sete negativas. Pode-se observar que ela é considerada útil em certos contextos, como em projetos complexos ou equipes homogêneas. Há resultados, porém, que mostram que a prática pode piorar o esforço e a produtividade.

Segundo os documentos obtidos com a revisão sistemática, as práticas refatoração e testes de aceitação tiveram uma boa avaliação. A prática de programação em pares, entretanto, apresentou resultados contraditórios. Isso pode ser um indício que sinalize a menor efetividade da prática segundo a percepção dos usuários na pesquisa online [Ambler 2007], e sua decorrente não utilização com outras práticas.

6 CONSIDERAÇÕES FINAIS

O *Framework* de Práticas Ágeis [Vilain 2007] apresenta um conjunto de práticas ágeis de diversos métodos sem considerar, no entanto, questões sobre a efetividade dessas práticas. O objetivo desse trabalho era abordar essa questão da agilidade das práticas, e para isso foram identificadas informações a respeito da adoção e da efetividade de cada prática do *framework*. Isso foi possível a partir do aproveitamento dos dados de uma pesquisa online [Ambler 2007] de grande alcance e participação a respeito do movimento ágil. Uma versão dessa abordagem já havia sido feita e publicada no Simpósio Brasileiro de Sistemas de Informação (SBSI) de 2011 [Schoepping 2011].

Para avaliar os relacionamentos entre práticas, foi utilizada a técnica de análise de agrupamentos que teve como função classificar os dados da pesquisa de forma que grupos de práticas fossem concebidos conforme a sua efetividade percebida pelos usuários. Todos esses grupos de práticas foram organizados, conforme sua efetividade, em uma árvore hierárquica ou dendograma.

A fim de validar a pertinência dos grupos formados, foi realizado um experimento em ambiente acadêmico abordando o uso de algumas práticas ágeis. As práticas selecionadas para o experimento faziam parte de um dos grupos mais efetivos, e também incluía uma prática não pertencente a este grupo. O grupo referido era um dos mais efetivos segundo o dendograma, formado por quatro práticas: desenvolvimento coletivo de código, integração contínua, refatoração e testes de aceitação. A outra prática, externa ao grupo, era a prática de programação em pares. O objetivo do experimento foi confrontar a efetividade do uso das práticas de refatoração e de testes de aceitação com o uso das práticas de refatoração e programação em pares. Os resultados, entretanto, foram influenciados por diversos fatores e por isso descartados. Uma revisão sistemática também foi realizada para apurar a efetividade dessas interações, mas informações muito relevantes não foram agregadas. Constatou-se, de qualquer modo, que a prática de programação em pares possui avaliações contraditórias.

A proposta desse trabalho difere dos trabalhos apresentados no capítulo 3 principalmente por esse avaliar as práticas ágeis individualmente e aqueles avaliarem o método ágil como um todo. Além disso, esse trabalho tem uma abordagem mais empírica, através dos dados de uma pesquisa de opinião, e tem como foco a avaliação do conjunto de práticas do *Framework* de Práticas Ágeis [Vilain 2007]. Em

[Qumer 2008], o *framework* proposto avalia diferentes métodos de desenvolvimento a partir de informações da literatura; a abordagem é de certa forma subjetiva e menos empírica. O trabalho [Datta 2006] propõe a definição de um índice formado por alguns domínios, que são valorados também subjetivamente, que fornecem uma avaliação mais abrangente do método. Em [Abrantes 2007] é proposta uma caracterização para os métodos ágeis através de uma revisão sistemática da literatura; apesar de o objetivo desse trabalho não ser o mesmo, a caracterização ajuda a definir o que deve ser avaliado em uma prática ágil. O trabalho [Parsons 2007] tem uma dinâmica parecida com esse, pois também utiliza resultados de uma pesquisa de opinião para avaliar a agilidade dos métodos ágeis; por utilizar uma pesquisa que tinha como foco a opinião sobre os métodos, poucas observações sobre as práticas ágeis são apresentadas.

A informação obtida com esse trabalho (efetividade, grupos, adoção) permite que a definição de um novo processo a partir do *framework* seja facilitada, pois haverá indícios sobre a agilidade de cada uma das práticas. A vantagem de ter acesso a esta informação facilita a revisão do processo definido a fim de eliminar eventuais práticas que poderiam comprometer a agilidade do processo e, consequentemente, as demandas do sistema de informação.

É importante considerar alguns aspectos acerca desse trabalho:

- O estudo realizado aqui foi baseado em dados de uma pesquisa online, que foram considerados verdadeiros. As pesquisas de opinião, entretanto, podem conter vícios que influenciem os resultados. De qualquer forma, elas costumam fornecer uma boa aproximação da realidade, e por esse motivo optou-se pelo aproveitamento da pesquisa [Ambler 2007].
- Um pequeno experimento foi realizado para confirmar as observações feitas através dos dados da pesquisa. Os resultados do experimento foram descartados pois foram influenciados por alguns fatores como, por exemplo, o tamanho do experimento e as práticas ágeis escolhidas.
- A revisão sistemática da literatura não trouxe resultados expressivos relacionados ao que se buscava e talvez não tenha utilizado os termos mais apropriados. Uma revisão com foco na efetividade de cada prática individualmente poderia ser interessante. A inclusão de mais bases de dados para a pesquisa também seria recomendado.

- Não foram consideradas algumas características que influenciam o desenvolvimento de software, como as naturezas de projeto, as políticas organizacionais e a cultura da equipe.

Como trabalho futuro, sugere-se a realização de um novo experimento, considerando os problemas e lições aprendidas durante o experimento realizado (APÊNDICE A). Para este novo experimento seria importante ter mais cuidado com as escolha das práticas, assim como na definição dos exercícios que irão testá-las. É importante também que haja homogeneidade entre os participantes do experimento; se esses fossem alunos, um treinamento acerca da prática ágil poderia ser uma solução. Abordar diferentes perfis, além do acadêmico, como o da indústria certamente enriqueceria os resultados. Uma revisão sistemática de todas as práticas também é interessante porque solidificaria os conhecimentos a respeito da efetividade de todas as práticas do *framework*.

Outro trabalho futuro, que já está em andamento, é o desenvolvimento de uma ferramenta web que facilite a definição de novos processos a partir do *framework* e que considere a análise da agilidade realizada nesse trabalho. Sugere-se que, além da análise realizada nesse trabalho, sejam considerados os princípios do movimento ágil. O novo processo definido poderia listar quais princípios ágeis serão abordados pelas práticas ágeis à medida que essas fossem adicionadas, visto que cada prática ágil está ligada a um ou mais princípios ágeis.

REFERÊNCIAS

[Abrahamsson 2002] Abrahamsson, Pekka. Salo, Outi. Ronkainen, Jussi. Warsta, Juhani. Agile software development methods: Review and analysis. Espoo, VTT Publications 478, 2002.

[Abrantes 2007] Abrantes, José Fortuna. Travassos, Guilherme Horta. Caracterização de Métodos Ágeis de Desenvolvimento de Software. VI Simpósio Brasileiro de Qualidade de Software, 2007.

[Agile Glossary] Agile Glossary. http://www.accurev.com/wiki/agile-glossary

[Ambler] Ambler, Scott W. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm

[Ambler 2006] Ambler, Scott W. Resultados da pesquisa Scott Ambler's March 2006 Agile Adoption Survey postado em http://www.ambysoft.com/surveys/

[Ambler 2007] Ambler, Scott W. Resultados da pesquisa Scott Ambler's March 2007 Agile Adoption Survey postado em http://www.ambysoft.com/surveys/

[Bassab 1990] Bassab, Wilton de Oliveira. Miazaki, Édina Shizue. Andrade, Dalton Francisco de Andrade. Introdução à Análise de Agrupamentos. IME-USP, 1990.

[Beck 1999] Beck, Kent. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.

[Beck 2004] Beck, Kent. Extreme Programming Explained: Embrace change, second edition. Addison-Wesley, 2004.

[Beck 2001] Beck, Kent. Beedle, Mike. Cockburn, Alistair. Fowler, Martin. et al. Manifesto for Agile Software Development. http://www.agilemanifesto.org/.

[Beck 2001b] Beck, Kent. Fowler, Martin. Planning Extreme Programming. Addison-Wesley, 2001.

[Canfora 2007] Canfora, Gerardo. Cimitile, Aniello. Garcia, Felix. Piattini, Mario. Visaggio, Corrado Aaron. Evaluating performances of pair designing in industry. Journal of Systems and Software, volume 80, páginas 1317–1327, 2007.

[Datta 2006] Datta, Subhajit. Agility measurement index: a metric for the crossroads of software development methodologies. ACM-44 SE, 2006.

[De Luca] De Luca, Jeff. Empresa de consultoria do criador do FDD. http://www.nebulon.com/fdd/index.html

[Fagundes 2005] Fagundes, Priscila Basto. *Framework* Para Comparação e Análise de Métodos Ágeis. Dissertação de Mestrado, Departamento de Informática e Estatística, UFSC, 2005.

[FDD] Site oficial da comunidade do Feature Driven Development. http://www.featuredrivendevelopment.com/

[Fowler] Fowler, Martin. http://martinfowler.com/articles/newMethodology.html#FromNothingToMonumentalToAgile

[Fowler 1999] Fowler, Martin. Beck, Kent. Brant, John. Opdyke, Willian. Roberts, Don. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

[Jacobson 2002] Jacobson, Ivar. A Resounding Yes to Agile Processes – But Also More. Cutter IT Journal, 2002.

[Haugset 2008] Haugset. Børge. Hanssen, Geir Kjetil. Automated Acceptance Testing: a Literature Review and an Industrial Case Study. Agile Conference, 2008.

[Khramov 2006] Khramov, Yuri. The Cost of Code Quality. Proceedings of the conference on AGILE, 2006.

[Kitchenham 2007] Kitchenham, Barbara. Charters, Stuart. Guidelines for performing Systematic Literature Reviews in Software Engineering, EBSE Techincal Report, versão 2.3, 2007.

[Lemos 2012] Lemos, Otávio Augusto Lazzarini. Ferrari, Fabiano Cutigi. Silveira, Fábio Fagundes. Garcia, Alessandro. Development of Auxiliary Functions: Should You Be Agile? An Empirical Assessment of Pair Programming and Test-First Programming. International Conference on Software Engineering (ICSE), 2012.

[Likert 1932] Likert, Rensis. A technique for the measurement of attitudes. Archives of Psychology, volume 22, número 140, 1932.

[Machado 2005] Machado, Thiago Leão. Uma Ferramenta de Suporte ao *Framework* para Comparação e Análise de Métodos Ágeis. Trabalho de Conclusão de Curso, Departamento de Informática e Estatística, UFSC, 2005.

[Melnick 2004] Melnik, Grigori. Maurer, Frank. Introducing Agile Methods: Three Years of Experience. Proceedings of the

30th EUROMICRO Conference, 2004.

[Muller 2001] Muller, Matthias M. Tichy, Walter F. Case study: extreme programming in a university environment. Proceedings of the 23rd International Conference on Software Engineering (ICSE), 2001.

[Parsons 2007] Parsons, David. Ryu, Hokyoung. Lal, Ramesh. The impact of methods and techniques on outcomes from agile software development projects. IFIP International Federation for Information Processing, volume 235/2007, páginas 235-249, 2007.

[Poppendieck 2003] Poppendieck, Mary. Poppendieck, Tom. Lean Software Development: An Agile Toolkit. Addison-Wesley, 2003.

[Pressman 2006] Pressman, Roger S. Engenharia de Software – 6. ed. McGraw-Hill, 2006.

[Qumer 2008] Qumer, Asif. Henderson-Sellers, Brian. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Information and Software Technology, 2008.

[R Project] The R Project for Statistical Computing. http://www.r-project.org.

[Schoepping 2011] Schoepping, Guilherme. Vilain, Patrícia. Analisando a Agilidade em Processos Ágeis. Simpósio Brasileiro de Sistemas de Informação (SBSI), 2011.

[Schwaber 2011] Schwaber, Ken. Sutherland, Jeff. Scrum Guide 2011. Disponível em: http://www.scrum.org/scrumguides/

[Scrum Alliance] Scrum Alliance. http://www.scrumalliance.org.

[Sfetsos 2010] Sfetsos, Panagiotis. Stamelos, Ioannis Empirical Studies on Quality in Agile Practices: A Systematic Literature Review. Seventh International Conference on the Quality of Information and Communications Technology, 2010.

[Sison 2007] Sison, Raymund. Yang, Theresa. Use of Agile Methods and Practices in the Philippines. Proceedings of the 14th Asia-Pacific Software Engineering Conference, 2007.

[Sommerville 2003] Sommerville, Ian. Software Engineering. Addison-Wesley, 2003.

[Usha 2009] Usha, K. Poonguzhali, N. Kavitha, E. A Quantitative Approach for Evaluating the Effectiveness of Refactoring in Software Development Process. International Conference on Methods and Models in Computer Science, 2009.

[Vanhanen 2005] Vanhanen, Jari. Lassenius, Casper. Effects of Pair Programming at the Development Team Level: An Experiment. International Symposium on Empirical Software Engineering, 2005.

[Vilain 2007] Vilain, Patrícia. Fagundes, Priscila B. Machado, Thiago L. A *Framework* for Selecting Agile Practices and Defining Agile Software Processes. SEKE, 2007.

[Wells] Wells, Don. Extreme Programming. http://www.extremeprogramming.org.

APÊNDICE A – Experimento

O objetivo do experimento é avaliar o impacto no desenvolvimento de software das práticas com maior nível de agrupamento com as práticas de menor nível de agrupamento. Dois aspectos foram considerados na avaliação deste impacto: corretude e esforço. Surge o interesse em responder as seguintes perguntas:

P1 – Usar Refatoração e Testes de aceitação ajuda a obter códigos com maior corretude do que usar Refatoração e Programação em pares?

P2 - Usar Refatoração e Testes de aceitação ajuda a obter códigos com menos esforço do que usar Refatoração e Programação em pares?

Destas duas perguntas derivam as hipóteses que deverão ser testadas através do experimento proposto:

Tabela 25. Hipóteses formuladas para o experimento

	Hipótese
H1	$\text{Corretude}_{RT} > \text{Corretude}_{RP}$
H2	$\text{Esforço}_{RT} < \text{Esforço}_{RP}$

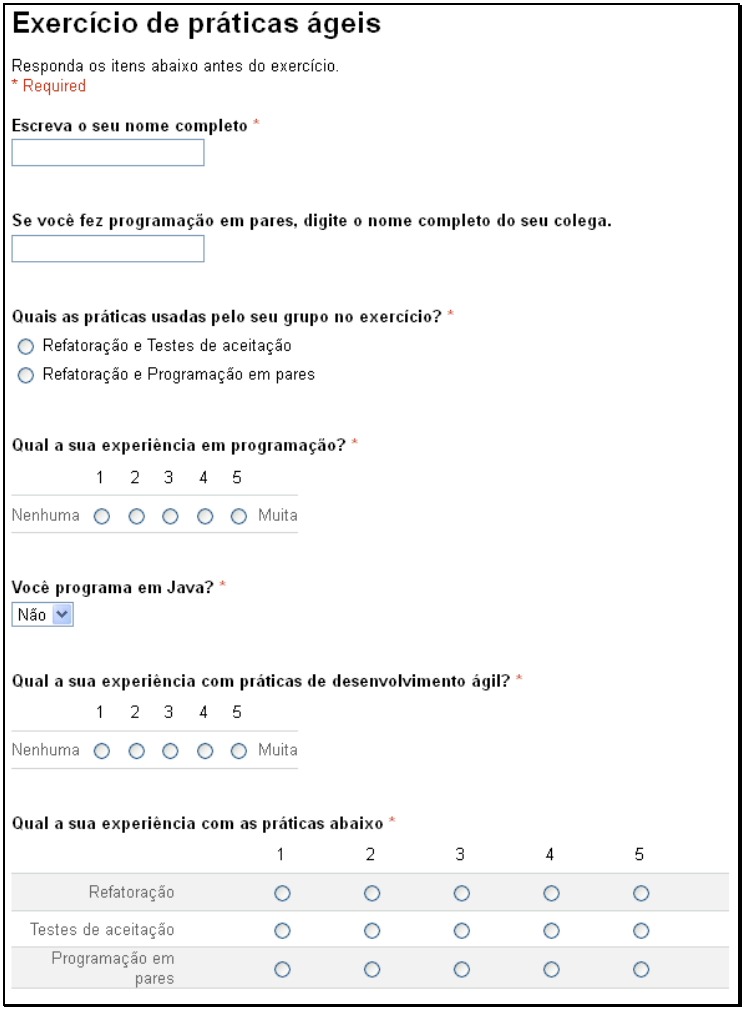
Legenda:

H = Hipótese;
RT = Refatoração e Testes de aceitação.
RP = Refatoração e Programação em pares.

Envolvidos

O experimento contou com a participação de 31 alunos de graduação do curso de Ciências da Computação, da Universidade Federal de Santa Catarina. Todos os alunos estavam matriculados na disciplina de Engenharia de Software I do curso, portanto, esperava-se certa homogeneidade quanto à experiência com programação e desenvolvimento ágil, mesmo que em um nível não avançado.

Isto pode ser confirmado através de uma pesquisa para levantar o perfil dos envolvidos no experimento, cujo formulário é mostrado abaixo.



Exercício de práticas ágeis

Responda os itens abaixo antes do exercício.
* Required

Escreva o seu nome completo *

Se você fez programação em pares, digite o nome completo do seu colega.

Quais as práticas usadas pelo seu grupo no exercício? *
- Refatoração e Testes de aceitação
- Refatoração e Programação em pares

Qual a sua experiência em programação? *

	1	2	3	4	5	
Nenhuma	○	○	○	○	○	Muita

Você programa em Java? *
Não

Qual a sua experiência com práticas de desenvolvimento ágil? *

	1	2	3	4	5	
Nenhuma	○	○	○	○	○	Muita

Qual a sua experiência com as práticas abaixo *

	1	2	3	4	5
Refatoração	○	○	○	○	○
Testes de aceitação	○	○	○	○	○
Programação em pares	○	○	○	○	○

A grande maioria informou possuir experiência com programação em nível intermediário e nenhum dos participantes informou possuir nenhuma experiência (Figura 13). Era importante também a familiaridade com a linguagem Java, posto que esta era a linguagem utilizada no exercício do experimento; 29 dos 31 alunos afirmaram programar em Java (Figura 14).

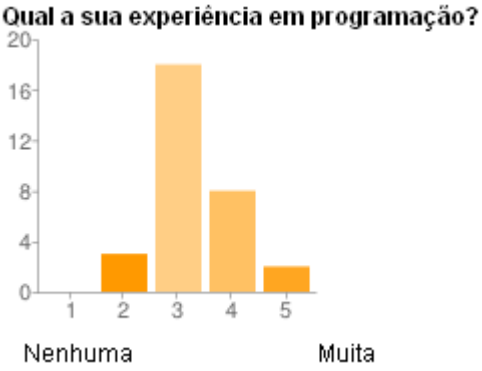


Figura 13. Pergunta 1 do questionário

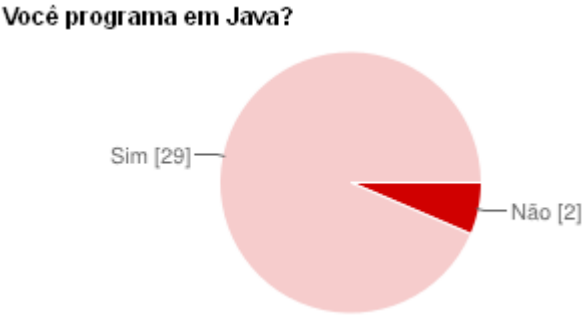


Figura 14. Pergunta 2 do questionário

Os métodos ágeis foram sumariamente apresentados durante a disciplina de Engenharia de Software I, assim como as práticas envolvidas neste experimento. Não se julgou necessário nenhum tipo de treinamento adicional. Foram incluídas na pesquisa, mesmo assim, perguntas para avaliar o conhecimento dos alunos acerca do desenvolvimento e práticas ágeis. Os resultados mostraram que a grande maioria possuía conhecimentos preliminares quanto ao desenvolvimento ágil, refatoração, testes de aceitação e programação em pares (Figura 15, Figura 16, Figura 17, Figura 18). Alguns alunos, contudo, apresentaram conhecimentos mais avançados nas práticas de refatoração e programação (Figura 16, Figura 18).

Figura 15. Pergunta 3 do questionário

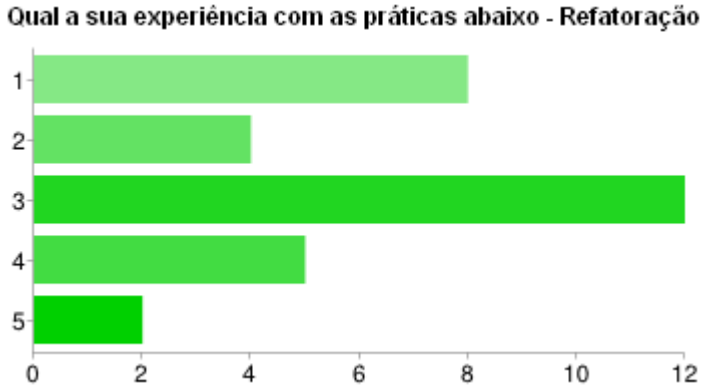


Figura 16. Pergunta 4 do questionário (1-Nenhuma, 5-Muita)

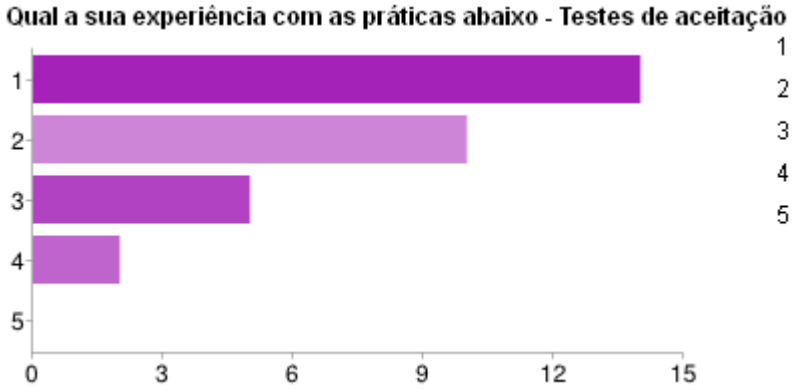


Figura 17. Pergunta 5 do questionário (1-Nenhuma, 5-Muita)

Figura 18. Pergunta 6 do questionário (1-Nenhuma, 5-Muita)

Independentemente dos perfis levantados, os alunos foram agrupados aleatoriamente em pares. Os pares, por sua vez, foram divididos entre os que usariam um par de práticas ou o outro, ou seja, oito pares usariam as práticas de Refatoração e de Testes de aceitação e outros oito pares usariam as práticas de Refatoração e Programação em pares.

Tabela 26. Dois grupos com diferentes práticas a serem usadas

Grupo A Alunos que usaram Refatoração e Testes de aceitação.
Grupo B Alunos que usaram Refatoração e Programação em pares.

No grupo A houve uma pequena divisão de responsabilidades. Enquanto um integrante da dupla tinha como principal atribuição a refatoração, o outro tinha a confecção dos testes de aceitação. No grupo B não houve esta divisão, pois estes deveriam fazer a refatoração e a programação em pares. Os 31 alunos foram divididos em dezesseis duplas, sendo oito pertencentes ao grupo A e oito ao grupo B. O número ímpar de alunos implicou em uma "dupla" com apenas um integrante no grupo A.

Exercício

Para avaliar as práticas ágeis selecionadas foi proposta a refatoração de um simples programa, que deveria ser auxiliada pelas práticas de Testes de aceitação ou de Programação em pares. O programa calcula o valor a ser pago pelo aluguel de filmes e o número de pontos de fidelidade adquiridos. Existem três tipos de filmes: infantil, acervo e lançamentos. O tipo do filme influencia no valor da locação e nos pontos de fidelidade acumulados. Este programa exemplo é o mesmo utilizado no livro sobre refatoração do autor Martin Fowler [Fowler 1999].

Os alunos receberam o código base do programa não refatorado e diagramas UML correspondentes (Figura 19, Figura 20). Também foram fornecidas instruções a respeito da refatoração (

ANEXO I), que tinha como foco o uso do polimorfismo e do padrão de projeto *State*. O ideal seria que os grupos refatorassem todo o código livremente, mas, em razão do curto espaço de tempo para o experimento, diagramas UML do código já refatorado (Figura 21, Figura 22) também foram fornecidos para orientar os alunos. Além da refatoração, algumas novas funcionalidades e alterações também deveriam ser implementadas:

1. Implementar uma interface gráfica ou em linha de comando;
2. Alterar o método de exibição das informações de locação;
3. Adicionar dois novos tipos de filme (Nacional e Promocional);
4. Alterar o cálculo dos filmes do tipo Lançamento.

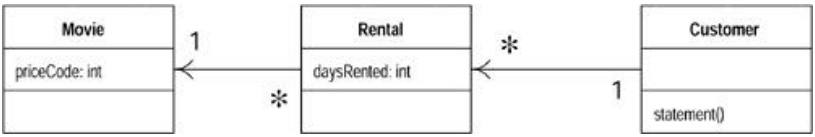


Figura 19. Diagrama de classes do programa não refatorado

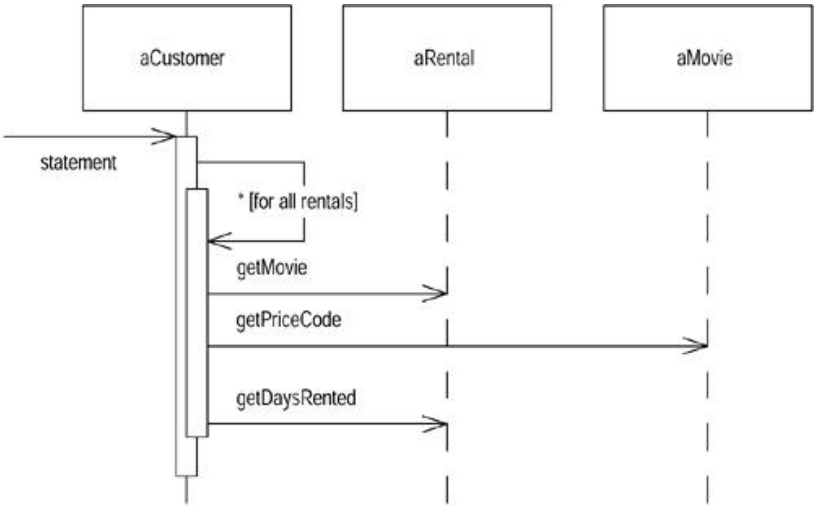


Figura 20. Diagrama de sequências do programa não refatorado

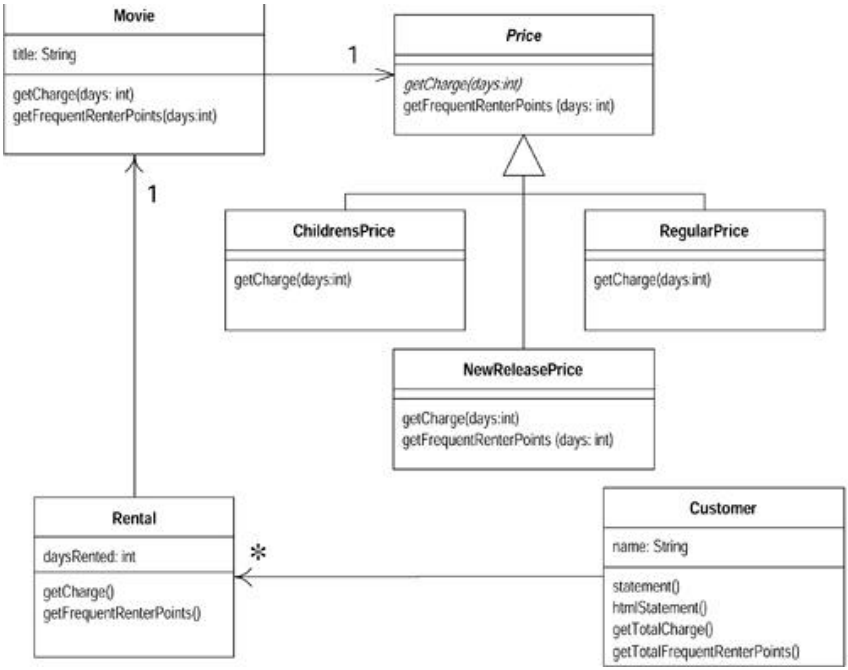


Figura 21. Diagrama de classes do programa refatorado

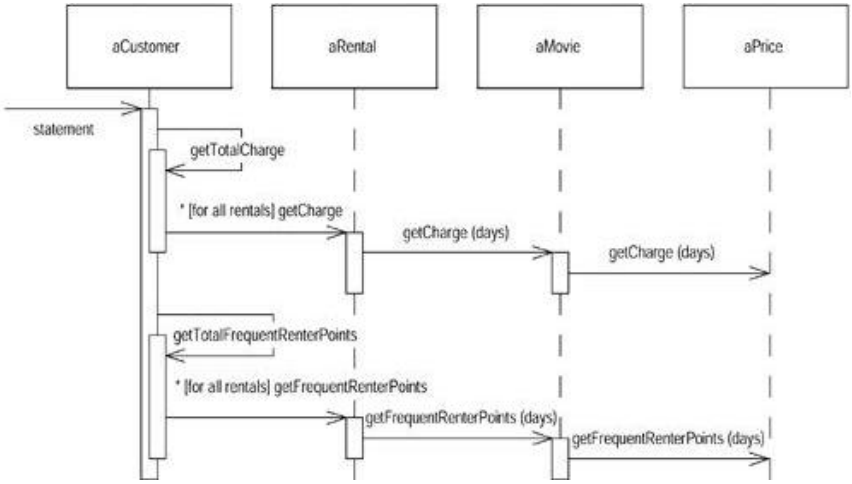


Figura 22. Diagrama de sequência do programa refatorado

Métricas

Os dois aspectos supracitados, corretude e esforço, foram avaliados através de duas métricas: métrica de confiabilidade e métrica de esforço. A definição dessas métricas está nas seções seguintes.

Métrica de confiabilidade

Para avaliar a confiabilidade do código produzido pelos grupos, foi definida a métrica de confiabilidade, que corresponde ao sucesso nos testes funcionais (STF). Ao todo, 60 testes funcionais foram implementados para testar os diferentes tipos de cálculo do programa. Esta métrica funciona em uma escala de três valores: B (baixa – quando o programa não passar em 15% dos testes), M (média – quando o programa passar entre 15 e 85% dos testes) e A (alta – quando o programa passar em mais de 85% dos testes). As três graduações funcionam de maneira simples; separam os códigos com pouquíssimos erros dos que apresentaram muitas falhas. Escalas semelhantes já foram utilizadas em outros trabalhos envolvendo práticas ágeis [Lemos 2012] [Canfora 2007].

Métrica de esforço

A outra métrica definida é referente ao esforço, que é simplesmente o tempo de desenvolvimento total (TDT) em minutos, também usada em [Lemos 2012]. O valor desta métrica corresponde ao tempo total que a dupla precisou para fazer o exercício. Como todos os alunos começaram o exercício no mesmo horário e com os mesmos recursos, apenas o horário de entrega do trabalho foi necessário para calcular esta métrica.

Resultados

A

Tabela *27* e a Tabela 28 mostram os valores obtidos para as métricas STF e TDT com o experimento. Percebe-se que houve pequena variação entre os grupos, certamente em decorrência do baixo número de duplas do experimento. De qualquer modo, pode-se observar que o grupo A (alunos que usaram refatoração e testes de aceitação) apresentou um código com menos número de falhas, 26 contra 29 do grupo B (alunos que usaram refatoração e programação em pares), ou seja, mais correto. Além disso, o número de duplas com STF igual a alto (A) do grupo A também foi superior. Em relação ao TDT, o grupo A

apresentou um valor minimamente menor (142 minutos contra 143 minutos do grupo B). Portanto, a respeito das hipóteses formuladas na

Tabela *25*, pode-se dizer, preliminarmente, que H1 ($Corretude_{RT}$ > $Corretude_{RP}$) é verdadeira e que H2 ($Esforço_{RT}$ < $Esforço_{RP}$) também é verdadeira.

Tabela 27. Resultados do experimento para o grupo A

Grupo A						
Dupla	**Início**	**Término**	**Falhas**	**Testes**	**TDT**	**STF**
A1	13:30	15:52	27	60	142	M
A2	13:30	16:11	1	60	161	A
A3	13:30	15:43	60	60	133	B
A4	13:30	16:25	3	60	175	A
A5	13:30	15:16	3	60	106	A
A6	13:30	15:54	28	60	144	M
A7	13:30	15:41	60	60	131	B
		Média	**26**	**-**	**142**	**-**

Tabela 28. Resultados do experimento para o grupo B

Grupo B						
Dupla	**Início**	**Término**	**Falhas**	**Testes**	**TDT**	**STF**
B1	13:30	15:30	59	60	120	B
B2	13:30	15:30	1	60	120	A
B3	13:30	15:59	30	60	149	M
B4	13:30	16:08	53	60	158	B
B5	13:30	16:00	3	60	150	A
B6	13:30	15:50	13	60	140	M
B7	13:30	16:13	41	60	163	M
		Média	**29**	**-**	**143**	**-**

Houve uma breve análise dos resultados do experimento e dos resultados do levantamento de perfil (apresentado na seção 5.1.1). Os grupos com melhores resultados no experimento não apresentaram um padrão de perfil. Aparentemente, o nível de conhecimento dos alunos a respeito de programação e práticas ágeis não influenciou os grupos com melhores resultados. Já os grupos com os piores resultados, tiveram, no levantamento de perfil, um padrão de resposta, com poucos conhecimentos acerca das práticas ágeis.

Os dados mostrados acima, todavia, não são suficientes para construir alguma inferência, pois os resultados podem ter sido influenciados pela amostra ou simplesmente pela coincidência.

Limitações

O experimento realizado possui algumas limitações. A divisão dos grupos e os diferentes níveis de experiência dos alunos podem influenciar os resultados de uma prática ou outra. A falta de experiência dos alunos com o uso das práticas ágeis também pode afetar negativamente os resultados. A pequena amostra, com poucas duplas de alunos, também dificulta a análise dos dados mais profundamente.

ANEXO I – Instruções para o exercício de práticas ágeis

Passo 1: Divisão dos grupos

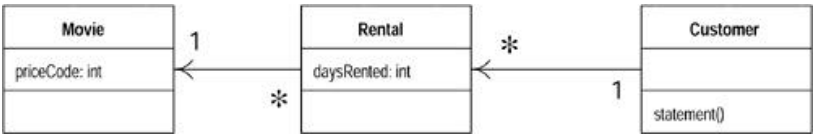
Todos deverão trabalhar em duplas e formarão dois grandes grupos, o grupo T1 e o grupo T2. O grupo T1 usará as práticas **Refatoração e Testes de aceitação**, e o grupo T2 usará as práticas **Refatoração e Programação em pares**. No grupo T1, enquanto uma pessoa faz a codificação/refatoração a outra faz os testes de aceitação que deverão ser executados quando concluída a refatoração. No grupo T2, os dois integrantes codificam/refatoram juntos.

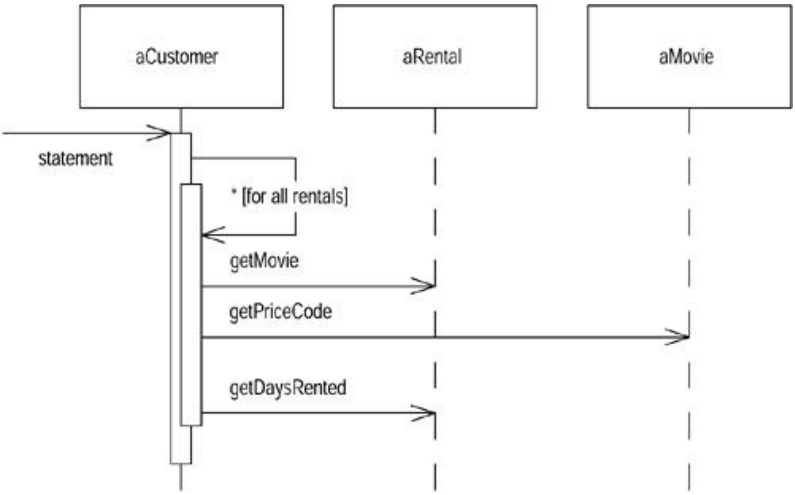
Passo 2: Resposta ao questionário

Antes de iniciar o exercício, todos deverão responder a este questionário:
https://docs.google.com/spreadsheet/viewform?formkey=dG1XNDlxSGktZWZVYW5nbFR0akxvYUE6MQ

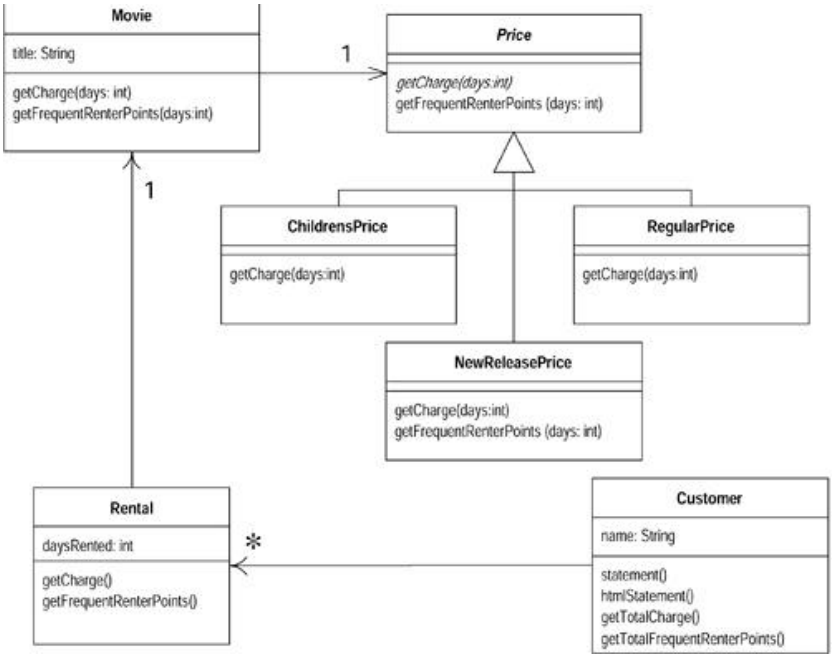
Passo 3: Refatoração

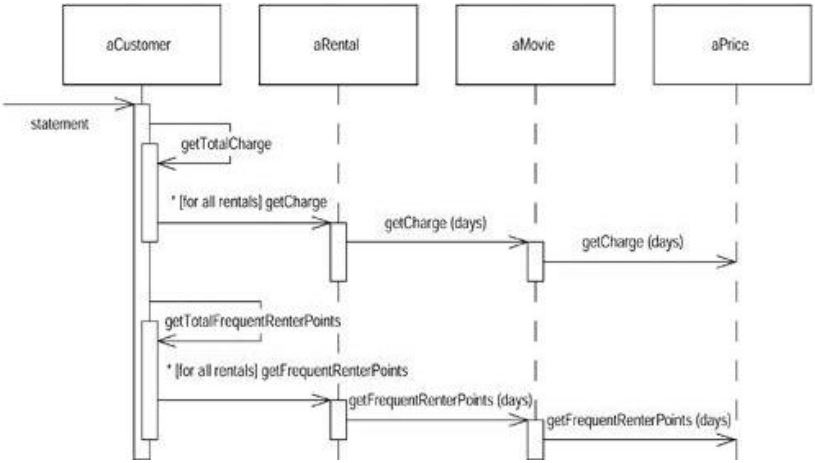
As duplas deverão refatorar o código "Example". A refatoração ocorrerá principalmente no método "statement", a fim de que este se torne mais coeso e faça uso de polimorfismo e do padrão de projeto State. Os diagramas abaixo orientarão a refatoração.
Código original:





Código refatorado:





Passo 4: Implementar novas funcionalidades

Após a refatoração do código, as seguintes mudanças deverão ser implementadas:

- Implementar menu textual (linha de comando) para adicionar Locação e listar Locações.
- Implementar um novo método, o "tableStatement" (substituir pelo htmlStatement do diagrama). Este deverá exibir as mesmas informações do método "statement" em formato tabela:

```
-------------------------------------------------
| Rental Record for                | Ana   |
-------------------------------------------------
| Ratatouille                      | 3.0   |
| Os Vingadores                    | 6.0   |
| Caça ao Terrorista               | 3.5   |
-------------------------------------------------
| Amount owed is                   | 12.5  |
-------------------------------------------------
| You earned 4 frequent renter points      |
-------------------------------------------------
```

- Adicionar uma nova classificação de filme (Promocional, 1 real/dia, 0 pontos de fidelidade)
- Adicionar uma nova classificação de filme (Nacional, 2 reais/dia, 2 pontos de fidelidade)
- Alterar cobrança dos filmes "NEW_RELEASE" (50% de desconto após o quarto dia)

Passo 5: Testes de aceitação

Este passo deve ser seguido apenas pelas duplas que fizerem parte do grupo T1, ou seja, que fizeram os testes de aceitação. Aqui eles deverão ser executados e seus resultados deverão ser documentados. É necessário que se faça ao menos **6 (seis) testes de aceitação** para este exercício. Os testes podem abordar a locação de cada um dos tipos de filmes e uma locação com diversos tipos; o valor cobrado e os pontos acumulados devem estar de acordo.

Passo 6: Entrega

No fim do exercício, todos deverão entregar o código fonte refatorado e os testes de aceitação, devidamente identificados. Também deverá ser anotada a **hora** em que ocorreu a entrega.

ANEXO II – Documentos encontrados na revisão sistemática

Lista dos documentos encontrados na revisão sistemática, excluídos os inválidos:

Moving from Waterfall to Agile	2008
7 Years of Agile Management	2008
A brief report on working smarter with Agile software development	2010
A cross-program investigation of students' perceptions of agile methods	2005
A cross-program investigation of students' perceptions of agile methods	2005
A *Framework* for Systematic Evaluation of Process Improvement Priorities	2011
A Model of Agile Evolution and Maintenance Process	2009
A ontology-based process modelling for XP	2003
A quantitative approach for evaluating the effectiveness of refactoring in software development process	2009
A Soft-Structured Agile *Framework* for Larger Scale Systems Development	2009
A Study of the Characteristics of Behaviour Driven Development	2011
A survey of evidence for test-driven development in academia	2008
A tentative *framework* for managing software product development in small companies	2002
Adopting Code Reviews for Agile Software Development	2010
Agile Implementation Phase in Two Canadian Organizations	2008
Agile methods applied to embedded firmware development	2004
Agile offshore techniques - a case study	2005
Agile practices in software development - experiences from student projects	2006
Agile software development in large organizations	2004
Agile Software Development Supporting Higher Education Reform	2007
Agile software testing in a large-scale project	2006
Agility counts in developing small-size software	2007

An agile boot camp: Using a LEGO-based active game to ground agile development principles	2011
An Agile Development Team's Quest for CMMI® Maturity Level 5	2009
An agile request for proposal (RFP) process	2003
An analysis of the history of classical software development and agile development	2009
An Analytical Survey of "On-Site Customer" Practice in Extreme Programming An Analytical Survey of "On-Site Customer" Practice in Extreme Programming	2008
An Empirical Study of the Relationship of Stability Metrics and the QMOOD Quality Models Over Software Developed Using Highly Iterative or Agile Software Processes	2007
An Era of Change-Tolerant Systems	2007
An experience in combining flexibility and control in a small company's software product development process	2003
Analyses of an agile methodology implementation	2004
Application of agile method in the enterprise website backstage management system: Practices for extreme programming	2012
Architectural design and documentation: Waste in agile development?	2012
Assessing quantitatively a programming course	2004
Automated Acceptance Testing Using Fit	2009
Automated Acceptance Testing: A Literature Review and an Industrial Case Study	2008
Balancing hands-on and research activities: a graduate level agile software development course	2005
Case study: extreme programming in a university environment	2001
Causal Factors, Benefits and Challenges of Test-Driven Development: Practitioner Perceptions	2011
Certifying for CMM Level 2 and IS09001 with XP@Scrum	2003
Colossal, Scattered, and Chaotic (Planning with a Large Distributed Team)	2008
Comparing Agile Software Processes Based on the Software Development Project Requirements	2008

Comparing Aspects with Conventional Techniques for Increasing Testability	2008
Defining Agile Software Quality Assurance	2006
Delivering software into NASA's Mission Control Center using agile development techniques	2012
Determining the Applicability of Agile Practices to Mission and Life-Critical Systems	2007
DevCOP: A Software Certificate Management System for Eclipse	2006
Developing Software Products for Mobile Markets: Need for Rethinking Development Models and Practices	2005
Development of a Weather Forecasting Code: A Case Study	2008
Double Trouble: Mixing Qualitative and Quantitative Methods in the Study of eXtreme Programmers	2004
Effects of Agile Methods on Website Quality for Electronic Commerce	2008
Effects of pair programming at the development team level: an experiment	2005
Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review	2011
Empirical Studies on Quality in Agile Practices: A Systematic Literature Review	2010
EMR standards in cardiological outpatient management	2004
Enhancing learning success in the introductory programming course	2003
Envisioning the Next-Generation of Functional Testing Tools	2007
Escape the waterfall: Agile for aerospace	2009
Evaluating Extreme Programming Effect through System Dynamics Modeling	2009
Evidence-based software process recovery: A post-doctoral view	2011
Evolving to a "lighter" software process: a case study	2001
Experiences with Extreme Programming in Telehealth: Developing and Implementing a Biosecurity Health Care Application	2005
Experiences with Using Assessment Based, Double-Loop Learning to Improve Engineering Student's Design Skills	2012

Experiment about test-first programming	2002
Exploring extreme programming in context: an industrial case study	2004
Exploring Synergistic Impact through Adventures in Group Pairing	2009
Extreme programming from a CMM perspective	2001
Factors Contributing to Software Quality Practices - An Australian Case Study	2007
Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review	2011
Factors that impact implementing an agile software development methodology	2007
Formal Versus Agile: Survival of the Fittest	2009
Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams	2009
Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams	2008
Genesis and Evolution of the Agile Movement in Brazil -- Perspective from Academia and Industry	2011
Global software development using agile methodologies: A *review* of literature	2012
Guest Editors' Introduction: TDD--The Art of Fearless Programming	2007
Guide to the Software Engineering Body of Knowledge 2004 Version	2004
Health Modelling for Agility in Safety-Critical Systems Development	2006
Hidden dependencies in software systems	2010
Human and social factors of software engineering: workshop summary	2005
Identifying risks in XP projects through process modelling	2006
Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study	2007
Improving agile software development using extreme AOCE and aspect-oriented CVS	2005

Improving software process in agile software development projects: results from two XP case studies	2004
Increasing Learning in an Agile Environment: Lessons Learned in an Agile Team	2011
Industrial experiences with automated regression testing of a legacy database application	2011
Influence of Large-Scale Organization Structures on Leadership Behaviors	2009
Integrating pair programming into a software development process	2001
Integrating usability engineering and agile software development: A literature *review*	2010
Introducing agile methods: three years of experience	2004
Introducing TDD on a free libre open source software project: a simulation experiment	2004
Knowledge repository to improve agile development processes learning	2010
Knowledge Support in Software Process Tailoring	2005
Maintenance and agile development: Challenges, opportunities and future directions	2009
Management challenges to implementing agile processes in traditional development organizations	2005
Measuring, monitoring and controlling software maintenance efforts	2006
New software development paradigms and possible adoption for security	2003
Novel fuzzy approach for ranking test vectors	2012
On the Effectiveness of Unit Test Automation at Microsoft	2009
Open Source Peer Review 2013; Lessons and Recommendations for Closed Source	2012
Patterns for agile development practice part 3 (version 4)	2006
Perceptions of extreme programming: a pilot study	2005
Perceptions of extreme programming: an exploratory study	2006
Potter Model - A Change Compliant Software Development Lifecycle Model	2011
Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model	2009

Process-centered *review* of object oriented software development methodologies	2008
Processes for software development within the Public Administration	2009
ReAjax: a reverse engineering tool for Ajax Web applications	2012
Reasoning about Faults in Aspect-Oriented Programs: A Metrics-Based Evaluation	2011
Refactoring as Testability Transformation	2011
Refactoring-Aware Configuration Management for Object-Oriented Programs	2007
Scaling Agile: Finding your Agile Tribe	2008
Self	2007
Self-Organizing Roles on Agile Software Development Teams	2012
Software Development as a Service: Agile Experiences	2011
Software Entropy in Agile Product Evolution	2010
Software Knowledge Capture and Acquisition: Tool Support for Agile Settings	2009
Some results of experimentation with extreme programming paradigm	2003
Structural Complexity and Programmer Team Task Strategy: An Experimental Test	2012
Subclassing XP: breaking its rules the right way	2004
Table of content	2009
Table of contents	2004
Teaching Computer Science Courses Using Extreme Programming (XP) Methodology	2005
Teaching software engineering and software project management: An integrated and practical approach	2012
Technical and human perspectives on pair programming	2004
Test case quality in test driven development: A study design and a pilot experiment	2012
Test Driven Design Methodology for Component-Based System	2007
Test-driven development concepts, taxonomy, and future direction	2005
The Agile Maturity Map A Goal Oriented Approach to Agile Improvement	2007

The cost of code quality	2006
The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis	2012
The impact of agile methods on software project management	2005
The Real World Software Process	2002
The Relationship between Developers and Customers in Agile Methodology	2008
The Software Engineering Timeline: A Time Management Perspective	2007
There has to be a better way! [software development]	2005
Thoughts on weak links and Alexandrian life in Scrum	2008
Towards empirical evaluation of test-driven development in a university environment	2003
TTCN-3 Test Data Analyser Using Constraint Programming	2008
Understanding Post-Adoptive Agile Usage -- An Exploratory Cross-Case Analysis	2011
UniX Process, Merging Unified Process and Extreme Programming to Benefit Software Development Practice	2009
Use of Agile Methods and Practices in the Philippines	2007
Using Bayesian Belief Networks to Model Software Project Management Antipatterns	2006
Using Factor Analysis to Generate Clusters of Agile Practices (A Guide for Agile Process Improvement)	2010
Using Faults-Slip-Through Metric as a Predictor of Fault-Proneness	2010
Using Scrum for Software Engineering Class Projects	2012
Using the Agile Unified Process in Banking	2010
Value based extreme programming	2006
What Makes Testing Work: Nine Case Studies of Software Development Teams	2009
What Types of Defects Are Really Discovered in Code Reviews?	2009
XP and Junior Developers: 7 Mistakes (and how to avoid them)	2007
YP and urban simulation: applying an agile programming methodology in a politically tempestuous domain	2003