Uso Eficaz de Métricas em Métodos Ágeis de Desenvolvimento de Software

Danilo Toshiaki Sato

DISSERTAÇÃO APRESENTADA
AO INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO
DO GRAU DE MESTRE EM CIÊNCIAS

Área de Concentração : **Ciência da Computação**
Orientador : **Prof. Dr. Alfredo Goldman vel Lejbman**

São Paulo, 29 de junho de 2007

Resumo

Os Métodos Ágeis surgiram no final da década passada como uma alternativa aos métodos tradicionais de desenvolvimento de software. Eles propõem uma nova abordagem para o desenvolvimento, eliminando gastos com documentação excessiva e burocrática, enfatizando a comunicação, colaboração com o cliente e as atividades que trazem valor imediato na criação de software com qualidade. Por meio de um processo empírico, com ciclos constantes de inspeção e adaptação, a equipe trabalha sempre num ambiente de melhoria contínua.

Uma das atividades propostas pela Programação Extrema (XP) para guiar a equipe em direção à melhoria é conhecida como *tracking*. O papel do *tracker* é coletar métricas para auxiliar a equipe a entender o andamento do projeto. Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples. Além disso, alguns dos possíveis problemas não são facilmente reconhecidos a partir de dados quantitativos.

Este trabalho investiga o uso de métricas no acompanhamento de projetos utilizando Métodos Ágeis de desenvolvimento de software. Um estudo de caso da aplicação de XP em sete projetos acadêmicos e governamentais foi conduzido para validar algumas dessas métricas e para avaliar o nível de aderência às práticas propostas, com o objetivo de auxiliar o *tracker* de uma equipe ágil.

Abstract

Agile Methods appeared at the end of the last decade as an alternative to traditional software development methods. They propose a new style of development, eliminating the cost of excessive and bureaucratic documentation, and emphasizing the interactions between people collaborating to achieve high productivity and deliver high-quality software. With an empirical process, based on constant inspect-and-adapt cycles, the team works towards an environment of continuous improvement.

One of the practices proposed by Extreme Programming (XP) to enable the creation of such environment is called tracking. The role of a tracker is to collect metrics that support the team on understanding their current process. Finding effective ways to evaluate the team and the development process is not an easy task. Also, some of the possible problems are not always easily recognizable from quantitative data alone.

This work investigates the use of metrics for tracking projects using Agile Methods for software development. A case study on the adoption of XP in seven academic and governmental projects was conducted to validate some of these metrics and to evaluate the level of adherence to the proposed practices, with the goal of assisting and supporting the tracker of an agile team.

Sumário

1. Introdução

1.1. Cenário

Software é um termo relativamente novo na Ciência da Computação. No início da década de 60, muitos fabricantes de *hardware* acreditavam que utilizar suas máquinas para automatizar tarefas administrativas seria algo direto e trivial, apesar da necessidade da programação [52]. O software não era visto como algo lucrativo, mas sim como algo distribuído gratuitamente pelos fabricantes de *hardware* ou escrito especificamente para uma instalação de computador [64].

Foi em 1968 e 1969, durante duas conferências organizadas pelo Comitê de Ciências da Organização do Tratado do Atlântico Norte (OTAN), que o termo *"engenharia de software"* ficou conhecido [80, 88]. Um dos objetivos dessas conferências foi discutir os problemas da indústria de software. O termo *"engenharia de software"* foi escolhido para enfatizar o quanto era necessário que a produção de software fosse baseada nos fundamentos teóricos e nas disciplinas práticas tradicionalmente conhecidos dos diversos ramos da engenharia [47]. Apesar de não chegarem a um consenso sobre a existência de uma crise do software, concordou-se que havia, no mínimo, sérios problemas.

Desde então, essa disciplina vem evoluindo. Técnicas para lidar com a complexidade do software foram surgindo e se sofisticando. Um dos primeiros processos de desenvolvimento de software surgiu em 1970 e ficou conhecido como processo em cascata. É comum citar Winston Royce como o autor do primeiro artigo que descreveu esse processo [92], no entanto ele foi mal interpretado [71]. Seu artigo dizia que um processo composto por fases distintas, desde o

levantamento de requisitos até a implantação, só serviria para os projetos mais simples. Ao invés disso, Royce dizia que um projeto deveria ser feito duas vezes quando houvesse fatores desconhecidos. Para um projeto de 30 meses sua recomendação era de fazer um projeto-piloto de 10 meses que seria jogado fora.

Os processos foram se aprimorando com o tempo. O modelo em espiral, proposto no meio da década de 80 [20], serviu como base para os atuais processos iterativos e incrementais como o *Rational Unified Process* (RUP) [59] e inclusive, mais recentemente, os Métodos Ágeis. Atualmente, a indústria busca modelos de certificação de qualidade de processo, como o *Capability Maturity Model Integration* (CMMI) da Carnegie Mellon [31] ou a ISO 9000-3 [68].

No entanto, apesar dos esforços para definir processos e métodos cada vez mais burocráticos e rigorosos, a *"crise do software"* não foi totalmente resolvida. Sistemas mal-construídos, software sem garantia e projetos fracassados são sintomas dessa crise [38]. Um estudo do Standish Group conhecido como CHAOS Report aponta alguns dos problemas da indústria de software entre 1994 [101] e 2003 [102]: a porcentagem de projetos não concluídos diminuiu de 31% para 15%; os projetos bem sucedidos aumentaram de 16% para 34%; o estouro médio dos custos diminuiu de 180% para 43%, enquanto o estouro médio do prazo diminuiu de 222% para 82%. Apesar de mostrar uma tendência de melhoria, os números ainda mostram uma série de problemas. Além disso, Brooks sugere que existe uma complexidade inerente à natureza do software que fará com que seu desenvolvimento seja sempre difícil e, em uma de suas publicações mais famosas, afirma que "não há uma bala de prata", ou seja, não existe uma única solução que resolva todos os problemas [28].

Apesar disso, a indústria de software cresceu e, com o surgimento da Internet, transformou-se em uma das indústrias mais importantes do mundo. Junto com essas mudanças, mudaram também as abordagens para desenvolvimento de software. Para atender às novas necessidades de negócio é preciso inovar e criar uma nova cultura de desenvolvimento de software [56].

A analogia entre a disciplina tradicional da engenharia e o desenvolvimento de software não

é muito adequada. Cockburn sugere que o desenvolvimento de software depende muito mais das pessoas e da comunicação [34]. Larman sugere que, ao contrário do cenário numa linha de produção em massa, o software não é algo previsível ou imune a mudanças. Desenvolver software é como desenvolver novos produtos [71]. Desenvolver é como criar uma receita, enquanto produzir é seguir os passos de uma receita [85]. O desenvolvimento é um processo de aprendizado, que envolve tentativas e erros. Como a manufatura previsível não pode ser comparada ao software, nenhuma das práticas e valores enraizados nesse paradigma podem trazer benefício [71].

Foi a partir desse novo paradigma que surgiram os **Métodos Ágeis de Desenvolvimento de Software**. Eles propõem uma nova abordagem de desenvolvimento, eliminando gastos com documentação excessiva e burocrática, enfatizando a interação entre as pessoas e nas atividades que efetivamente trazem valor e produzem software com qualidade [13].

Dentre os Métodos Ágeis, destaca-se a Programação Extrema, ou XP, criada a partir de diversas práticas de sucesso adotadas na indústria e formalizada em 1999 por Kent Beck, com a publicação do livro *"Extreme Programming Explained: Embrace Change"*. A Programação Extrema propõe um conjunto de valores, princípios e práticas, que visam garantir o sucesso no desenvolvimento de software, em face a requisitos vagos e que mudam constantemente [9, 12].

Em XP os valores são parte integral do processo e sua adoção exige mudanças culturais e comportamentais em relação às abordagens tradicionais [62]. Dentre outras características, Robinson e Sharp sugerem que, para o sucesso na adoção de XP, uma equipe deve encorajar ativamente a preservação da qualidade no trabalho do dia-a-dia e deve reavaliar e reafirmar seus objetivos constantemente [90, 91].

Uma das atividades propostas em XP para guiar a equipe em direção à melhoria e mostrar eventuais problemas com o processo é conhecida como *tracking*. Kent Beck descreve o papel do *tracker* em uma equipe de XP como alguém responsável por coletar freqüentemente métricas a partir de dados obtidos com a equipe e por garantir que o time compreenda o que está

sendo medido [9, 12]. Nem sempre é fácil para o *tracker* decidir quais informações coletar e quais apresentar para a equipe. Além disso, alguns dos possíveis problemas não são facilmente reconhecidos a partir de dados quantitativos.

Atualmente, os Métodos Ágeis são utilizados em diversos contextos, desde pequenas, médias e grandes empresas até agências governamentais e universidades, para construir uma variedade de sistemas e aplicações de software. Sua adoção tem mostrado excelentes resultados que estão amplamente descritos na literatura [27, 46, 74, 73, 79, 94].

1.2. Objetivos

O principal objetivo deste trabalho é realizar um estudo sobre métricas para auxiliar o acompanhamento (*tracking*) de projetos utilizando Métodos Ágeis de desenvolvimento de software. Um estudo de caso da aplicação de XP em sete projetos de diferentes contextos será conduzido para validar algumas das métricas apresentadas e avaliar o nível de aderência às práticas propostas. Por meio da colaboração com os *trackers* de cada equipe, métricas quantitativas e qualitativas serão coletadas a partir de diversas fontes, como: código-fonte, repositório de código, entrevistas e questionários. Essas métricas serão posteriormente avaliadas e analisadas.

Por fim, este trabalho classificará os sete projetos em relação aos termos e categorias propostos pelo *Extreme Programming Evaluation Framework* (XP-EF) [112, 113]. Esse arcabouço descreve o contexto do estudo de caso, a extensão da adoção das práticas de XP e os resultados dessa adoção, contribuindo para a construção de uma base de evidências empíricas da adoção dos Métodos Ágeis e da Programação Extrema.

1.3. Organização do Trabalho

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta em mais detalhes os Métodos Ágeis e, particularmente, a Programação Extrema. O Capítulo 3 discute o papel

das métricas e do *tracker* num projeto ágil. No Capítulo 4, um estudo de caso com sete projetos de diferentes contextos é apresentado, classificando-os no XP-EF, discutindo e analisando a evolução de algumas métricas. Por fim, o Capítulo 5 conclui este trabalho, destacando as contribuições e apresentando possibilidades para trabalhos futuros. Dentre as contribuições, o Apêndice A apresenta um questionário de aderência às práticas de XP e o Apêndice B apresenta um catálogo de medidas para auxiliar o *tracker* a escolher as métricas de acompanhamento mais apropriadas para algumas das práticas de XP.

2. Métodos Ágeis

Nos últimos anos, os Métodos Ágeis de desenvolvimento de software vêem ganhando importância em diversos segmentos da indústria de software. Assim como os métodos tradicionais, os Métodos Ágeis têm por objetivo construir sistemas de alta qualidade que atendam às necessidades dos usuários. A principal diferença está nos princípios utilizados para atingir tal objetivo.

Os Métodos Ágeis apresentam uma abordagem bastante pragmática para o desenvolvimento de software. Planos detalhados são feitos apenas para a fase atual do projeto. Para fases futuras, os planos são considerados apenas rascunhos que podem se adaptar a mudanças conforme o time aprende e passa a conhecer melhor o sistema e as tecnologias utilizadas.

Este capítulo apresenta algumas evidências que motivaram o surgimento dos Métodos Ágeis, explicando seus valores e princípios, com ênfase na Programação Extrema, um dos Métodos Ágeis mais utilizados na indústria.

2.1. Evidências

No desenvolvimento de software, é comum que os requisitos mudem enquanto a implementação ainda está acontecendo. Kajko-Mattson et al. mostram que cerca de 40% a 90% do custo durante o ciclo de vida de um projeto é gasto na fase de manutenção [67]. Muitas empresas e times de desenvolvimento acham que mudanças são indesejáveis pois acabam com todo o esforço gasto no planejamento. No entanto, os requisitos geralmente mudam conforme

o cliente vê o sistema sendo implantado e em funcionamento. É muito difícil criar um plano no início do projeto que consiga prever todas as mudanças sem gastar muito esforço, tempo e dinheiro.

Boehm disse que "encontrar e arrumar um defeito no software após a entrega custa cerca de 100 vezes mais do que encontrá-lo e arrumá-lo nas fases iniciais de design" [21]. Essa foi uma das principais justificativas para os métodos tradicionais gastarem mais tempo nas fases de análise de requisitos e design[1], apesar do próprio Boehm ter sugerido o desenvolvimento iterativo ao invés da "produção do produto completo de uma vez" [23]. Em 2001, num artigo de Boehm e Basili, houve uma redução no pessimismo ao perceber que, para sistemas pequenos, o fator era mais próximo de 5 : 1 ao invés de 100 : 1 e que, mesmo para sistemas grandes, boas práticas arquiteturais poderiam reduzir de forma significativa o custo da mudança, encapsulando as áreas de mudança em partes pequenas e fatoradas [22].

Poppendieck [85] sugere que a principal razão das mudanças no desenvolvimento de software é que o processo de negócio ao qual o software está atrelado evolui constantemente. Construir flexibilidade para acomodar mudanças arbitrárias é muito caro e pode ser um desperdício. Segundo Johnson [63], 45% das funcionalidades implementadas num sistema típico não são utilizadas nunca e 19% são raramente utilizadas. A melhor estratégia é evitar generalizações desnecessárias e fazer com que o sistema seja flexível apenas nas áreas mais propícias à mudança [86].

A maioria dos estudos de caso na indústria apontam para uma taxa relativamente alta de fracassos nos projetos de software [65]. No clássico relatório do Standish Group de 1994, o CHAOS Report [101], 37% dos fatores relacionados aos projetos em dificuldade estavam relacionados aos requisitos. O mesmo relatório de 2003 aponta que apenas 52% das funcionalidades são entregues em um projeto [102]. Outro estudo de classificação de defeitos aponta os requisitos como principal categoria de defeitos, com 41% [100]. "Nós queremos estabilizar os

[1] Optamos por deixar a palavra "design" sem destaque (itálico) por já ser bem consolidada e conhecida na comunidade técnica

requisitos, mas eles não são estáveis" [71].

2.2. O Manifesto Ágil

Em fevereiro de 2001, um grupo formado por 17 desenvolvedores experientes, consultores e líderes da comunidade de software se reuniu em Utah para discutir idéias e procurar uma alternativa aos processos burocráticos e às práticas adotadas nas abordagens tradicionais da Engenharia de Software e gerência de projetos. Dessa reunião surgiu o *Manifesto do Desenvolvimento Ágil de Software* [13], que destaca as diferenças com relação às abordagens tradicionais e define seus valores:

- **Indivíduos e interações** são mais importantes que processos e ferramentas;
- **Software funcionando** é mais importante que documentação completa e detalhada;
- **Colaboração com o cliente** é mais importante que negociação de contratos;
- **Adaptação a mudanças** é mais importante que seguir um plano.

Apesar da importância dos itens à direita, os Métodos Ágeis dão mais valor para os itens destacados à esquerda. Além dos quatro valores básicos, o Manifesto Ágil também apresenta 12 princípios que auxiliam a difusão de suas idéias:

- A maior prioridade é a satisfação do cliente através da entrega rápida e contínua de software que traga valor;
- Mudanças nos requisitos são aceitas, mesmo em estágios avançados de desenvolvimento. Processos ágeis aceitam mudanças que trarão vantagem competitiva para o cliente;
- Software que funciona é entregue freqüentemente, em períodos que variam de semanas a meses, quanto menor o tempo entre uma entrega e outra melhor;

- As pessoas relacionadas ao negócio e os desenvolvedores devem trabalhar juntos no dia a dia do projeto;
- Construa projetos formados por indivíduos motivados, fornecendo o ambiente e o suporte necessário e confiando que realizarão o trabalho;
- O modo mais eficiente e eficaz de transmitir informações dentro e fora do time de desenvolvimento é a comunicação face a face;
- A principal medida de progresso é software funcionando;
- Processos ágeis promovem o desenvolvimento sustentável. Os investidores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante por tempo indefinido;
- Cuidar continuamente da excelência técnica e do bom design ajuda a aprimorar a agilidade;
- Simplicidade – a arte de maximizar a quantidade de trabalho não necessário – é essencial;
- Os melhores requisitos, arquiteturas e design surgem de equipes auto-gerenciadas;
- Em intervalos regulares, o time reflete sobre como se tornar mais eficiente, refinando e ajustando seu comportamento apropriadamente.

Essas características trazem dinamismo para o desenvolvimento, motivação para o time e informações mais precisas sobre a verdadeira situação do projeto para o cliente. Enquanto as abordagens tradicionais têm um enfoque mais preditivo, os Métodos Ágeis são adaptativos.

O Manifesto Ágil apresenta uma nova filosofia para o desenvolvimento de software. Sob seus valores e princípios aparecem diversas abordagens mais específicas, com diferentes idéias, comunidades e líderes. Cada comunidade forma um grupo distinto, porém todas seguindo os mesmos princípios. É comum inclusive a troca de conhecimento e práticas entre membros de diferentes comunidades, formando um ecosistema em torno de diversos métodos, detalhados a seguir com maior ênfase na Programação Extrema.

2.3. Scrum

Desenvolvida nas décadas de 80 e 90 por Ken Schwaber, Jeff Sutherland, e Mike Beedle [98, 97]. O Scrum se concentra mais nos aspectos gerenciais do desenvolvimento de software, através de iterações de duas semanas ou 30 dias (chamados *sprints*) com monitoramento diário através de reuniões em pé (ou *stand-up meetings*). Por ter menos ênfase nos aspectos técnicos, é geralmente combinada com práticas propostas por XP e compatível com certificações de qualidade como CMMI ou ISO-9001 [107].

2.4. Lean Software Development

Com base no Sistema de Produção da Toyota [82], o movimento *Lean* revolucionou a manufatura e, mais recentemente, o desenvolvimento de produtos e o gerenciamento da cadeia de suprimentos (*supply chain management*). Mary e Tom Poppendieck traçaram os paralelos entre os valores e práticas *Lean* com o desenvolvimento de software, fornecendo sete princípios [85, 86]: "Elimine Desperdícios", "Inclua a Qualidade no Processo", "Crie Conhecimento", "Adie Comprometimentos", "Entregue Rápido", "Respeite as Pessoas" e "Otimize o Todo".

2.5. Família Crystal

Alistair Cockburn propõe uma família de métodos por acreditar que diferentes abordagens são necessárias para equipes de tamanhos diferentes [33]. Apesar disso, todos os métodos dessa família compartilham propriedades como: entrega freqüente, reflexão e comunicação. Outra parte importante dos métodos da família Crystal é o que Cockburn chama de habitabilidade (*habitability*): o mínimo de processo necessário para que a equipe consiga ter sucesso.

2.6. Feature Driven Development (FDD)

Desenvolvida por Peter Coad e Jeff de Luca no final da década de 90, a FDD define 2 fases compostas por 5 processos bem definidos e integrados: a fase de concepção e planejamento, composta por "desenvolver um modelo abrangente", "construir uma lista de funcionalidades" e "planejar por funcionalidade"; e a fase iterativa de construção, composta por "detalhar por funcionalidade" e "construir por funcionalidade" [84]. Outra característica interessante da FDD é a utilização de modelos UML em cores para representar classes com diferentes responsabilidades (chamados "arquétipos") [32].

2.7. Adaptive Software Development

Proposto por Jim Highsmith, esse método tenta explorar a natureza adaptativa e a incerteza no desenvolvimento de software [55]. Com base nas idéias dos sistemas adaptativos complexos (também conhecidos como Teoria do Caos), ele propõe 3 fases não-lineares e possivelmente sobrepostas: especulação (referindo-se ao paradoxo do planejamento), colaboração e aprendizado. Por meio de iterações curtas, a equipe cria o conhecimento cometendo pequenas falhas, causadas por falsas premissas, e corrigindo-as aos poucos, criando uma experiência mais rica e ampla.

2.8. Dynamic System Development Method (DSDM)

Desenvolvido inicialmente por um consórcio de empresas britânicas em 1994, esse método se baseou nas idéias do desenvolvimento rápido de aplicações (*Rapid Application Development* ou RAD) e no desenvolvimento iterativo e incremental [103]. O método começa com 2 fases: um estudo de viabilidade que valida se o processo DSDM é apropriado para o projeto e um estudo de negócio, para entender as necessidades de negócio e definir uma arquitetura e os requisitos iniciais. O resto do processo é formado por 3 fases: uma iteração para o modelo

funcional define protótipos e uma documentação de análise inicial, uma iteração para o design e construção do sistema, e uma última fase de implementação para entrega e implantação do produto. O DSDM ainda define princípios que incluem: participação ativa do usuário, entrega freqüente, times com poder de decisão e testes durante todo o ciclo de vida do produto.

2.9. Programação Extrema

A Programação Extrema (*Extreme Programming* ou XP) foi um dos Métodos Ágeis que mais recebeu atenção na virada do século. Seu objetivo é a excelência no desenvolvimento de software, visando baixo custo, poucos defeitos, alta produtividade e alto retorno de investimento. Na segunda edição do livro *"Extreme Programming Explained"* [12], Kent Beck aprimora a definição de XP da primeira edição, enumerando suas principais características:

- **XP é um método leve**. O time só deve fazer o necessário para trazer valor para o cliente;
- **XP é uma método que enfatiza o desenvolvimento de software**. Apesar de ter implicações em áreas como *marketing*, vendas ou operações, XP não tenta resolver os problemas diretamente ligados a elas;
- **XP funciona para times de qualquer tamanho**. Apesar das práticas de XP funcionarem melhor em times pequenos, seus valores e princípios podem ser aplicados em qualquer escala;
- **XP se adapta a requisitos vagos ou em constante mudança**.

2.9.1. Histórico

As idéias de XP se originaram de conversas entre Kent Beck e Ward Cunningham a partir de suas experiências com desenvolvimento de software em Smalltalk. Juntos, eles escreveram

o primeiro artigo sobre cartões CRC [15] e sobre a aplicação de padrões no desenvolvimento de software [14] Ward Cunningham foi o criador do Wiki [72] e foi no seu Wiki que as primeiras discussões sobre XP aconteceram. Muitas das características de XP, como Refatoração, Programação Pareada, adaptação à mudança, Integração Contínua, desenvolvimento iterativo e ênfase nos testes, são elementos-chave presentes na cultura da comunidade Smalltalk desde a década de 1980.

Em 1992, William Opdyke publicou sua tese de doutorado [83], contando como Kent e Ward obtinham ganhos em produtividade usando técnicas de Refatoração. Mais tarde, essas técnicas seriam compiladas no livro de Martin Fowler et al. sobre refatoração [45].

Kent Beck também publicou o primeiro artigo sobre testes de unidade automatizados [8] quando desenvolveu o primeiro arcabouço para desenvolvimento de testes automatizados: o SUnit [7] para testes em Smalltalk e, mais tarde juntamente com Erich Gamma, o JUnit [18,17] para testes em Java. Esse arcabouço foi portado para diversas linguagens, formando um conjunto de ferramentas que ficaram conhecidas como família XUnit: JUnit (Java), CppUnit (C++), CUnit (C), NUnit (.NET), pyUnit (Python), Test::Unit (Ruby), dentre diversos outros [114].

Todas essas idéias foram se fundindo na cabeça de Kent Beck quando, em 1996, ele foi chamado para ajudar no projeto C3, ou *Chrysler Comprehensive Compensation System*, que ficou conhecido como o berço de XP. Nele, Kent Beck utilizou pela primeira vez todas as práticas que vieram a se tornar a Programação Extrema. Sua idéia, que originou o nome Programação Extrema, era juntar as boas práticas de programação já conhecidas pela indústria e encará-las como botões de volume que seriam aumentados ao valor máximo, extremo. Se fazer revisão de código era bom, fazer Programação Pareada era revisão em tempo integral; se fazer testes automatizados era bom, escrevê-los antes do código era melhor ainda.

Foi a partir dessa experiência que Kent Beck lançou, em 1999, a primeira edição do livro que difundiu XP, *"Extreme Programming Explained: Embrace Change"* [9]. Esse livro recebeu no

mesmo ano o prêmio JOLT de produtividade da revista *Software Development* e, após cinco anos de experiência com a utilização e consultoria de XP, Kent Beck lançou, em parceria com a sua esposa, da área de psicologia, a segunda edição do livro que é hoje uma das principais referências sobre o assunto [12].

2.9.2. Abordagem

Segundo Kent Beck [12], a Programação Extrema inclui:

- Uma filosofia para o desenvolvimento de software baseada nos valores de comunicação, *feedback*, simplicidade, coragem e respeito;
- Um conjunto de práticas comprovadamente úteis para melhorar o desenvolvimento de software. As práticas expressam os valores de XP;
- Um conjunto complementar de princípios, técnicas intelectuais que auxiliam a tradução dos valores em práticas, úteis quando as práticas existentes não resolvem seu problema particular;
- Uma comunidade que compartilha os mesmos valores e muitas das mesmas práticas.

Essa separação entre valores, princípios e práticas já estava presente na primeira edição de XP, porém sua importância foi reforçada na segunda edição. É possível ter uma visão mais ampla do processo quando pensamos nessas três perspectivas.

As práticas são técnicas utilizadas no dia-a-dia dos membros de uma equipe de XP. Elas são claras, objetivas e específicas. Práticas como Desenvolvimento Dirigido por Testes (*Test Driven Development* ou TDD) [10] ou Refatoração [45] fazem sentido somente no contexto da programação. Em outro contexto as mesmas práticas não fariam sentido. As práticas de XP são apresentadas na Seção 2.9.5 e 2.9.6, que compara as diferenças entre as abordagens na primeira e na segunda edição.

Valores são critérios mais amplos e universais utilizados para julgar uma determinada situação. É possível enxergar o valor do *feedback* em diversos contextos, desde a programação (através da Integração Contínua, por exemplo) até a comunicação com o cliente (através do Ciclo Semanal e do Ciclo Trimestral). XP é uma disciplina de software baseada nos seguintes valores: comunicação, simplicidade, *feedback*, coragem e respeito [61]. Os valores são discutidos na Seção 2.9.3.

Os valores dão razão às práticas, enquanto as práticas evidenciam os valores. Para preencher o espaço vazio entre valores e práticas, Kent Beck apresenta os princípios de XP. Os princípios são técnicas intelectuais para auxiliar na tradução de valores em práticas. Eles devem ser utilizados quando as práticas propostas não se aplicam numa situação específica. Por exemplo, o princípio dos Passos Pequenos é demonstrado em diferentes práticas, como a Implantação Diária ou o ritmo imposto pelo Desenvolvimento Dirigido por Testes. Os princípios de XP são apresentados na Seção 2.9.4.

2.9.3. Valores

A filosofia de trabalho proposta por XP está baseada em 5 valores que servem de base para a aplicação das práticas e dos princípios:

Comunicação

O primeiro valor do Manifesto Ágil propõe que os indivíduos e as interações entre eles são importantes para o desenvolvimento de software [13]. A Programação Extrema expressa tal importância através do valor da Comunicação. XP pressupõe que a maioria dos problemas num projeto de software ocorrem por dificuldade na comunicação [71]. A Comunicação é evidenciada em muitas das práticas: uma equipe de XP funciona como um Time Completo, trabalhando numa Área de Trabalho Informativa na qual todos possam Sentar Junto. A presença e o Envolvimento Real com o Cliente e a Programação Pareada também são práticas

que fortalecem e facilitam a comunicação.

Simplicidade

O segundo valor de XP é a simplicidade. Os membros de uma equipe de XP estão freqüentemente buscando a solução mais simples para resolver seus problemas atuais. Num contexto onde a adaptação a mudanças é aceita [13] e encorajada, XP promove a preocupação com o que é mais simples para resolver os problemas de hoje, evitando desperdícios com soluções genéricas para modificações futuras.

Segundo Kent Beck, a simplicidade é o valor intelectual mais intenso de XP [12]. Encontrar a solução mais simples não é uma tarefa fácil. A Seção 2.9.5 apresenta uma das práticas que mais ajuda os desenvolvedores numa equipe de XP a manter a ênfase constante no design simples, o Desenvolvimento Dirigido por Testes [10] .

Feedback

É importante ter uma resposta rápida sobre as ações realizadas para se adaptar às mudanças [13]. XP promove ciclos curtos e constantes de *feedback*, nos mais variados aspectos do desenvolvimento de software. Os valores de XP se complementam, por isso o *feedback* é parte importante da comunicação e da simplicidade. Diante de uma dúvida entre três diferentes soluções, tentar todas parece ser um desperdício, porém esta pode ser a melhor forma de descobrir qual solução é mais simples e mais fácil de lidar.

Segundo Ambler [2], a maior contribuição para o sucesso dos Métodos Ágeis é a redução dos ciclos de *feedback*. As práticas de XP evidenciam tal valor: a Programação Pareada fornece *feedback* em segundos enquanto o Desenvolvimento Dirigido por Testes fornece *feedback* em minutos.

Coragem

O quarto valor de XP é a coragem. Kent Beck descreve a coragem em XP como a "ação tomada diante do medo" [12]. Isso não significa que os membros da equipe devem ter coragem para fazer o que quiserem sem se preocupar com as conseqüências para o time. A coragem como valor primário, sem a influência e balanceamento dos outros valores, pode ser perigosa. No entanto, em conjunto com os outros valores, ela é muito poderosa.

Teles [106] enumera alguns pontos onde a adoção de XP exige coragem da equipe para: desenvolver software de forma incremental; manter a ênfase constante na simplicidade; permitir ao cliente priorizar funcionalidades; incentivar os desenvolvedores a trabalhar em par; investir tempo em refatoração; investir tempo em testes automatizados; estimar as histórias na presença do cliente; compartilhar o código com todos os membros da equipe; fazer a integração completa do sistema diversas vezes ao dia; adotar um ritmo sustentável; abrir mão da documentação; propor contratos de escopo variável e propor a adoção de um processo novo.

Respeito

O quinto valor, enfatizado na segunda edição de XP, serve de base para os outros quatro valores: respeito. Se os membros da equipe não se importam com os outros e/ou com os resultados, XP não vai funcionar [12]. É importante reconhecer que a excelência no desenvolvimento de software depende das pessoas, e elas devem se respeitar para conseguir extrair o máximo de seu potencial.

Algum dos pontos na adoção de XP que podem ser influenciados pela falta de respeito são: comunicação sem respeito criará conflitos internos; coragem sem respeito trará atitudes que vão contra o bem estar da equipe; Programação Pareada é um exercício contínuo de respeito; horas-extras excessivas irão impactar o ritmo sustentável da equipe; a colaboração entre a equipe e o cliente também exige uma comunicação aberta e respeitosa.

2.9.4. Princípios

Os princípios de XP funcionam como ferramentas para tradução dos valores em práticas. Tanto documentos longos quanto conversas diárias têm a intenção de comunicar. Descobrir qual é a forma mais eficiente depende parte do contexto e parte dos princípios intelectuais. Nesse caso, o princípio da Humanidade sugere que a conversa satisfaz melhor as necessidades humanas de relacionamento. Os princípios que guiam a Programação Extrema são:

Humanidade

Pessoas desenvolvem software. É importante levar em conta as necessidades básicas do ser humano no desenvolvimento de software, criando oportunidades para: crescimento, contribuição, participação e relacionamento. O grande desafio é balancear as necessidades pessoais com as necessidades do time. As práticas de XP tentam atender tanto às necessidades de negócio quanto às necessidades pessoais dos membros da equipe.

Economia

Os envolvidos no desenvolvimento de software também devem se preocupar com os aspectos econômicos para evitar que o projeto seja apenas um "sucesso técnico". É importante que uma equipe de XP esteja constantemente preocupada em agregar valor de negócio ao sistema que estão desenvolvendo. Esse princípio é um dos motivos pelos quais os clientes são responsáveis pela priorização das histórias nas reuniões de planejamento. A equipe de XP deve resolver os problemas mais importantes primeiro, maximizando o valor do projeto.

Benefício Mútuo

O princípio do Benefício Mútuo é um dos mais importantes de XP, porém é também um dos mais difíceis de aplicar. Todas as atividades devem trazer benefício a todos os envolvidos. Escrever documentos longos é um exemplo de violação desse princípio: o programador diminui

seu ritmo de produção para escrever um documento que não tem valor agora, podendo apenas trazer valor no futuro para alguém que irá dar manutenção no seu código (se a documentação continuar válida até lá). XP resolve o problema da comunicação com o futuro de uma forma mutuamente benéfica através de:

- Testes automatizados que ajudam o programador no design e na implementação das funcionalidades agora, servindo como documentação e como teste de regressão para as pessoas que irão dar manutenção no futuro;
- Refatoração constante para remover complexidade desnecessária, simplificar o design e remover defeitos, deixando o código mais limpo e fácil de entender para os futuros mantenedores;
- Nomes coerentes e baseados em metáforas que facilitam o entendimento do sistema, aumentando a velocidade atual do desenvolvimento e da integração de novos programadores na equipe.

Auto-Semelhança

O princípio da Auto-Semelhança sugere a aplicação da estrutura de uma solução em outros contextos, inclusive em diferentes escalas. Um exemplo sugerido por Kent Beck [12] é a aplicação de testes a priori em ambos os níveis: não só quando desenvolvendo os testes unitários com TDD, mas também para especificar os testes de aceitação com o cliente. Dessa forma, fica mais claro para os programadores que as histórias só estão prontas quando passam nos testes de aceitação, reduzindo o ciclo de *feedback* e simplificando o andamento da iteração.

Melhoria

XP valoriza a busca constante pela melhoria. Ao invés de buscar a perfeição, é mais importante tentar fazer o melhor trabalho possível hoje, e estar consciente de tudo que será necessário

para melhorar amanhã. O princípio da Melhoria valoriza as atividades que começam agora e se refinam ao longo do tempo.

Diversidade

Os times devem ser formados por uma variedade de habilidades, atitudes e perspectivas. Dessa diversidade podem surgir conflitos que devem ser vistos como oportunidades para discussão das diferentes perspectivas. Muitas vezes o melhor design surge a partir de soluções distintas. O princípio da Diversidade se expressa em XP através da prática do Time Completo.

Reflexão

Uma boa equipe não deve apenas fazer o seu trabalho, mas sim pensar constantemente sobre as razões e as formas como estão trabalhando. Os times devem analisar seus sucessos e suas falhas, sempre em busca da melhoria contínua. A reflexão vem após a ação. O aprendizado surge para a equipe como resultado da reflexão sobre a ação. Para maximizar o *feedback*, as equipes de XP devem refletir e estar conscientes de seus atos.

Fluxo

Esse princípio sugere a entrega de um fluxo contínuo de software que agrega valor ao negócio, evitando pensar em fases discretas. Quanto maior o tamanho de uma atividade – iteração, *release*, história ou tarefa – maior o tempo gasto até descobrir se ela foi realizada com sucesso ou não, o que aumenta o risco do erro. Quanto maior o erro, mais difícil é a correção. Para aumentar o *feedback* e diminuir os riscos, a equipe de XP deve prover incrementos pequenos de funcionalidade, fazendo entregas pequenas e freqüentes. Práticas como Integração Contínua, *Build* em 10 Minutos, Implantação Incremental e Implantação Diária evidenciam o princípio do Fluxo.

Oportunidade

Equipes de XP enxergam problemas como oportunidades para mudança. Na busca pela excelência, os membros do time devem demonstrar uma atitude positiva, identificando oportunidades para aprender, melhorar e desenvolver software de qualidade.

Redundância

O princípio da Redundância sugere que os problemas difíceis e críticos devem ser resolvidos de várias maneiras diferentes. Muitas das práticas de XP, como Programação Pareada, Integração Contínua, Envolvimento Real com o Cliente e Desenvolvimento Dirigido por Testes, são redundantes na tentativa de reduzir a quantidade de defeitos e aumentar a qualidade do software produzido.

Falha

Complementando o princípio da Redundância, o princípio da Falha sugere que todo erro é um aprendizado. Na dúvida entre três soluções diferentes, tente implementar todas, mesmo que algumas falhem. Diante de diferentes opções de design, ao invés de perder tempo discutindo qual a melhor solução, vale mais a pena implementá-las paralelamente, aprendendo com os erros e chegando num consenso através da experiência.

Qualidade

Sacrificar a qualidade nunca é um meio eficaz de controle. Os projetos não andam mais rápido aceitando baixa qualidade. Cada incremento na qualidade tem reflexos de melhoria em diversas outras áreas do projeto, como produtividade, eficiência e motivação. Um projeto XP não considera a qualidade como uma das variáveis de controle. O custo e o tempo também são geralmente fixos, deixando o escopo como principal variável na negociação com o cliente [16].

Passos Pequenos

O princípio dos Passos Pequenos complementa o princípio do Fluxo, fazendo com que os membros da equipe de XP se perguntem sempre "Qual o mínimo que preciso fazer para garantir que estou na direção certa?". Práticas como Desenvolvimento Dirigido por Testes e Integração Contínua evidenciam o valor do ritmo imposto pelos passos pequenos. A Refatoração é outro exemplo da aplicação desse princípio.

Aceitação da Responsabilidade

A responsabilidade não deve ser imposta, deve ser aceita. As práticas refletem esse princípio ao, por exemplo, sugerir que a pessoa responsável por uma história também é responsável pela sua estimativa, design, implementação e teste.

2.9.5. Práticas

A abordagem de apresentação das práticas de XP foi totalmente refatorada na segunda edição por Kent Beck [12]. Ao invés de exigir a utilização de todas as 12 práticas de uma vez, Kent Beck sugere que cada time deve se adaptar da maneira que achar mais apropriada. Ao invés de impor as práticas, cada mudança deve começar pelos próprios membros da equipe. Além disso, as práticas na segunda edição são 24, divididas entre *práticas primárias* e *práticas corolárias*. *Práticas primárias* são aquelas que podem ser aplicadas separadamente, trazendo melhoria imediata para a equipe. *Práticas corolárias* são mais difíceis de implementar, mostrando sua eficiência somente após domínio e experiência prévia com as práticas primárias.

Práticas Primárias

- **Sentar Junto**: A equipe de XP deve trabalhar num espaço amplo e aberto, onde todos possam ficar juntos, fortalecendo os elos da comunicação. É preferível trocar os

tradicionais cubículos por áreas de uso comum onde os membros da equipe possam se agrupar para discutir no quadro branco, sentar juntos para trabalhar em par e espalhar gráficos e informações na parede.

- **Time Completo**: Equipes de XP devem ser multi-disciplinares, com todas as habilidades necessárias para o sucesso do projeto. A equipe deve ser formada não apenas por desenvolvedores, mas também por analistas de teste, analistas de negócio, clientes, especialistas em banco de dados, especialistas em interfaces gráficas, administradores de rede, etc. Todos devem trabalhar num espírito de contribuição para a equipe, visando o bom andamento do projeto.

- **Área de Trabalho Informativa**: Transforme o ambiente de trabalho num reflexo do projeto. Um observador interessado deve ser capaz de ter uma idéia da evolução do projeto apenas andando pela área de trabalho. Alguns exemplos de instrumentos para espalhar essas informações, chamados de radiadores de informação por Cockburn [34], são: cartões com histórias num mural, quadros brancos, notas em papel nas paredes e gráficos como, por exemplo, o *burn-down chart* proposto pelo Scrum [98, 97] para acompanhar a velocidade da equipe.

- **Trabalho Energizado**: O ritmo de trabalho não deve afetar a vida pessoal dos membros da equipe. Durante o planejamento, o número de horas dedicadas ao projeto deve ser definido realisticamente. Fazer horas-extra deve ser exceção e não a regra. Os membros da equipe de XP devem trabalhar apenas enquanto puderem ser produtivos e se manter energizados. O desenvolvimento de software exige criatividade que raramente aparecerá em momentos de cansaço ou indisposição [39].

- **Programação Pareada**: Os desenvolvedores trabalham em par para realizar suas tarefas. Isso promove o trabalho coletivo e colaborativo, une a equipe e melhora a comunicação e a qualidade do código. Os pares devem ser trocados regularmente, inclusive

várias vezes por dia. Geralmente, a seleção dos pares depende da tarefa a ser realizada, da disponibilidade dos membros da equipe e da experiência de cada um. O objetivo principal é espalhar o conhecimento do sistema pela equipe inteira. Como importante efeito colateral temos também o compartilhamento de técnicas e competências entre os membros da equipe.

- **Histórias**: O planejamento em XP é feito com histórias escritas em pequenos cartões. Cada cartão é escrito pelo cliente e deve descrever uma unidade de funcionalidade, que geralmente representa um requisito funcional desejado. Mike Cohn propõe o seguinte formato para uma história [35]:

  > "Como um <`usuário/papel`>
  > Eu gostaria de <`funcionalidade`>
  > Para que <`valor de negócio`>"

  Este formato é interessante pois evidencia o valor de negócio associado a cada funcionalidade. Para cada história, os desenvolvedores devem dar uma estimativa sobre o tempo para implementá-la e os clientes determinam a prioridade de cada história. Essas informações são utilizadas no Jogo do Planejamento, que acontece no início dos ciclos semanais e trimestrais. A descrição no cartão não deve armazenar todas as informações sobre a história. Os desenvolvedores de uma equipe de XP utilizam o diálogo como principal meio de comunicação com o cliente para elucidar dúvidas sobre os detalhes da história. Os cartões devem servir apenas como um lembrete desse diálogo.

- **Ciclo Semanal**: O software em XP é produzido de forma iterativa e incremental. Essa prática sugere que uma equipe de XP deve planejar o trabalho de cada iteração uma semana por vez. A cada semana, os membros da equipe se reunem para: refletir sobre o progresso realizado até o momento, planejar e priorizar as histórias da semana com o cliente e quebrar cada história em tarefas que serão implementadas pelos pares durante

a semana (Jogo do Planejamento).

- **Ciclo Trimestral**: As *releases* são planejadas a cada trimestre. O plano do trimestre é de mais alto nível, geralmente representado por um tema. Temas são diferentes de histórias pois, ao invés de se preocupar com os detalhes, abrangem o todo: a forma em que o projeto se encaixa na organização. Durante o planejamento do trimestre, o time deve: identificar gargalos (principalmente externos à equipe), iniciar reparos e escolher as histórias mais alinhadas ao tema e que serão implementadas durante o trimestre.

- **Folga**: Inclua no plano algumas tarefas menores que possam ser removidas caso ocorra um atraso. Estimativas não devem ser consideradas um comprometimento, pois geralmente são feitas com base na experiência pessoal de cada desenvolvedor, estando sujeitas a erros. No entanto, é importante que o time se comprometa com as entregas para o cliente. Para acomodar o caráter subjetivo das estimativas, um tempo de folga deve ser incluído no plano, para que eventuais atrasos não atrapalhem a entrega da iteração ou da *release*, criando um vínculo de confiança e responsabilidade entre a equipe e o cliente.

- ***Build* em 10 Minutos**: O *build* **automático** do sistema **inteiro** e a **bateria completa de testes** deve rodar em até **10 minutos**. Os itens em destaque são importantes: o *build*, assim como todas as tarefas repetitivas do projeto, deve ser automatizado, deve considerar o sistema inteiro (código-fonte e configurações de ambiente), deve rodar todos os testes e deve ser rápido o suficiente. As equipe de XP devem tentar atingir o máximo dos objetivos propostos por essa prática, pois quanto mais tempo o *build* demorar, menos será executado, diminuindo os ciclos de *feedback* e aumentando o tempo entre a introdução e a descoberta de um erro.

- **Integração Contínua**: O código-fonte fica armazenado num repositório compartilhado e cada par deve integrar suas alterações ao final de cada tarefa, após garantir que tudo está funcionando, realizando um *build* completo e rodando todos os testes. Os desen-

volvedores interagem com o repositório diversas vezes por dia para trabalhar sempre numa versão atualizada do sistema. Dessa forma, o conhecimento do sistema se espalha por toda a equipe mais facilmente e a dificuldade de realizar uma integração grande se dilui em diversas integrações pequenas e freqüentes.

- **Desenvolvimento Dirigido por Testes**: Um dos aspectos mais importantes de XP é a ênfase nos testes automatizados. Essa prática sugere que os testes sejam escritos antes do código, trazendo benefícios como: ênfase no desenvolvimento (evitando generalizações desnecessárias), preocupação com acoplamento e coesão (geralmente o design está com problemas quando surge uma dificuldade para escrever o teste), confiança (o teste verifica o comportamento agora e no futuro) e ritmo (a próxima tarefa é sempre escrever o próximo teste ou fazer o teste passar, criando o ritmo conhecido como "vermelho, verde e refatoração" [10]).

- **Design Incremental**: A simplicidade é um conceito chave que permite a adaptação a mudanças. Para minimizar o custo com mudanças desnecessárias no futuro, os desenvolvedores devem sempre implementar o design mais simples – e não o mais simplista – com o mínimo da complexidade e flexibilidade necessária para atender às necessidades de negócio atuais. Porém, deve-se tomar cuidado com a interpretação dessa prática. Seu objetivo não é minimizar o investimento com design no curto prazo, mas sim manter esse investimento proporcional às necessidades do sistema conforme ele evolui. O Design Incremental deve ter suporte de outras práticas, como a Refatoração e os testes automatizados gerados pelo Desenvolvimento Dirigido por Testes para garantir que a equipe seja capaz de solucionar os problemas futuros com rapidez.

Práticas Corolárias

- **Envolvimento Real com o Cliente**: Faça com que as pessoas cujas vidas e negócios serão afetados pelo sistema façam parte da equipe. O cliente também deve fazer parte do

Time Completo. Ele deve entender as necessidades de negócio e conhecer os verdadeiros usuários do sistema, para escrever histórias, definir prioridades e testes de aceitação e responder eventuais dúvidas sobre as funcionalidades desejadas.

- **Implantação Incremental**: Ao substituir um sistema legado, evite fazê-lo de uma só vez. É mais seguro substituir gradualmente partes das funcionalidades, deixando os dois sistemas funcionando ao mesmo tempo. Grandes implantações são muito arriscadas e têm custos humanos e econômicos muito altos [12].

- **Continuidade da Equipe**: Mantenha equipes eficientes trabalhando juntas. Existe uma tendência de tratar as pessoas como recursos substituíveis, que são trocadas de projetos diversas vezes para manter a utilização alta. No entanto, o valor no desenvolvimento de software não surge apenas do que as pessoas sabem ou fazem, mas também de seus relacionamentos e conquistas em equipe. Ignorar o valor das interações e relações para simplificar problemas de alocação é uma falsa economia [12].

- **Diminuição da Equipe**: Conforme a equipe melhora sua capacidade de produção, gradualmente reduza a carga sobre um dos membros, mantendo os outros trabalhando normalmente. Conforme a carga diminui sobre esse membro, ele pode ser liberado para formar novas equipes. Apesar do próprio Kent Beck não ter tido experiências com essa prática [12], incluiu-a em XP com base na sua eficácia no Sistema de Produção da Toyota [82, 85, 86]. Tentar fazer com que todos os membros pareçam ocupados pode possivelmente esconder um excesso de recursos na equipe.

- **Análise de Causa Inicial**: Sempre que um defeito for encontrado, conserte o problema e suas causas. O objetivo não é apenas fazer com que esse defeito específico nunca mais aconteça, mas também que o time nunca mais cometa o mesmo erro em outras situações. O processo de XP para consertar um defeito é: escrever um teste de aceitação automatizado que demonstre o problema, assim como o comportamento esperado; escrever um

teste unitário com o menor escopo que também reproduz o defeito; corrigir o sistema, fazendo todos os testes passarem e, por fim, tentar descobrir a causa inicial do defeito não ter sido detectado anteriormente, realizando as mudanças necessárias para evitar que o erro aconteça novamente.

- **Código Compartilhado**: O repositório do código-fonte é compartilhado por toda a equipe e qualquer um pode fazer melhorias em qualquer parte do sistema. Ao invés de identificar responsáveis por cada parte do código, o time inteiro é responsável pelo sistema inteiro. Com isso, os membros da equipe adquirem uma ampla visão do sistema, facilitando a execução de refatorações e espalhando o conhecimento por toda a equipe.

- **Código e Testes**: Os únicos artefatos mantidos pela equipe são o código e os testes. Documentação deve ser evitada e, caso estritamente necessária, deve ser gerada a partir do código e dos testes. A principal forma de comunicação em uma equipe de XP é a conversa. Artefatos que se tornem obsoletos com o tempo não agregam valor ao sistema e ao negócio. Eliminar desperdícios permite melhorar as áreas que agregam valor, aquelas que definem o que o sistema faz hoje e o que a equipe pode fazer com o sistema amanhã.

- **Repositório de Código Unificado**: A equipe deve desenvolver em um repositório único. Ramificações podem existir, mas devem ser evitadas. Quanto maior o número de versões concorrentes do mesmo código, maior o trabalho de sincronização e mais difícil é o entendimento pela equipe. Linhas paralelas devem ser integradas rapidamente e os motivos de sua existência devem ser reconsiderados constantemente e não tidos como verdade absoluta.

- **Implantação Diária**: Coloque novas versões do sistema em produção toda noite. Dessa forma, o ciclo de *feedback* entre o que está sendo feito pelo programador e o que está sendo utilizado pelo usuário é sempre rápido e eficiente. Para que essa prática seja eficaz, muitas outras devem estar funcionando bem. É preciso garantir que o número de defeitos

seja baixo e que as ferramentas de *build* e implantação automatizem todo o processo de entrega, possibilitando inclusive voltar uma versão, caso necessário.

- **Contrato de Escopo Negociável**: Contratos devem fixar tempo, custo e qualidade, deixando o escopo preciso aberto para negociação. As equipes de XP se adaptam a mudanças, permitindo que o cliente faça correções no escopo do software conforme seu aprendizado do sistema evolui. Em XP, o escopo é revisado freqüentemente para garantir que a equipe está sempre trabalhando no que é mais importante para o cliente.

- **Pague-Pelo-Uso**: Essa prática sugere a utilização do dinheiro como *feedback* final. Em sistemas *pay-per-use*, você cobra a cada vez que o sistema é utilizado. Conectar o fluxo econômico ao desenvolvimento de software provê informações precisas e atualizadas para direcionar melhorias no sistema. No modelo mais utilizado pela indústria, o cliente paga a cada *release*, porém isso coloca os interesses da equipe de desenvolvimento e do cliente em conflito. Enquanto a equipe deseja um número maior de *releases* com pouca funcionalidade, o cliente deseja o menor número possível de *releases* contendo o máximo de funcionalidade. Essa tensão gera problemas de comunicação e *feedback*.

2.9.6. Comparação com as Práticas da Primeira Versão

Devido à mudança na abordagem de apresentação das práticas de XP, Kent Beck transformou as 12 práticas originais em 24 práticas divididas em práticas primárias e práticas corolárias, conforme descrito na Seção 2.9.5. Algumas práticas da primeira versão ainda aparecem de forma subjetiva na descrição das novas práticas, mas para ter uma melhor base de comparação, é importante descrevê-las como foram apresentadas na primeira edição do livro [9]:

- **Refatoração**: A refatoração é uma técnica sistemática para reestruturar o código existente, alterando sua estrutura interna, porém mantendo seu comportamento externo [45].

Alguns exemplos de refatorações são: a remoção de código duplicado, a mudança do nome de um método ou variável e a extração de um trecho de código para um método auxiliar. O objetivo é sempre tornar o código e o design mais simples, legível, limpo e preparado para mudanças.

- **Metáfora**: Todos os membros da equipe, incluindo programadores e clientes, devem conversar sobre o sistema numa linguagem comum. Essa linguagem deve ser entendida tanto pelas pessoas técnicas, quanto pelas pessoas de negócio. Isso pode ser obtido através de uma metáfora comum que relaciona abstrações do sistema com objetos de um certo domínio, existentes no mundo real. Essa é uma das práticas mais difíceis de introduzir em uma equipe inexperiente pois está diretamente ligada à comunicação e ao modo como as pessoas estão dispostas a compartilhar seus desejos e suas idéias. Essa prática estava bastante alinhada com um padrão descrito por Ward Cunningham, que ficou conhecido como "Sistema de Nomes" [37]. Mais recentemente, o uso dessa linguagem ubíqua para representar conceitos do domínio no código-fonte ficou popularizada com a técnica de modelagem definida por Eric Evans, conhecida como Design Dirigido pelo Domínio (*Domain Driven Design*) [42].

- **Padronização de Código**: Antes de dar início à implementação, o time define um conjunto de padrões de codificação para escrita do código do sistema. Isso torna o código homogêneo e mais fácil de entender, melhora a comunicação, facilita a refatoração e promove a propriedade coletiva do código.

A Tabela 2.1 é uma adaptação do autor com base num artigo de Michele Marchesi [75] e apresenta uma comparação entre a primeira e a segunda edição de XP, destacando como as novas práticas se relacionam às práticas originais. As práticas entre parênteses são aquelas que não aparecem explicitamente na nova edição de XP.

Tabela de Comparação entre as práticas de XP [2]	
Primeira Edição	**Segunda Edição**
Jogo do Planejamento	Histórias, Ciclo Semanal, Ciclo Trimestral e Folga
Versões Pequenas	Ciclo Semanal, Implantação Incremental e Implantação Diária
Design Simples	Design Incremental
(Refatoração)	Design Incremental
Propriedade Coletiva do Código	Código Compartilhado e Repositório de Código Unificado
Ritmo Sustentável	Trabalho Energizado e Folga
Cliente com os Desenvolvedores	Time Completo e Envolvimento Real com o Cliente
(Metáfora)	Design Incremental
(Padronização de Código)	Código Compartilhado

Tabela 2.1.: Tabela de comparação entre as práticas da primeira e da segunda edição de XP, adaptado de [75]

2.9.7. Adaptações das Práticas de XP

A adoção de um Método Ágil como XP não depende simplesmente da aplicação direta das práticas. O ciclo empírico de inspeção e adaptação dos Métodos Ágeis sugere que as equipes estejam em busca freqüente de melhoria e algumas adaptações são permitidas. Conforme XP começou a ser utilizada em mais projetos, uma nova prática foi sugerida: "Conserte XP quando ela falha" [110]. Uma boa fonte de inspiração para tais adaptações são as práticas sugeridas em outros Métodos Ágeis [95]. Nos projetos que serão apresentados no Capítulo 4, duas práticas adaptadas foram utilizadas:

- **Reuniões em Pé**: prática utilizada em Scrum [98, 97] que consiste numa reunião informal curta e diária, realizada no início do dia de trabalho na qual cada membro da equipe responde à três perguntas: "O que fez ontem?", "O que pretende fazer hoje?" e "Quais problemas impedem o seu progresso?". Os membros da equipe participam dessa reunião em pé para garantir que sua duração seja curta. Além disso, eventuais problemas que

[2] As práticas a seguir não mudaram e não foram incluídas na tabela: **Programação Pareada**, **Integração Contínua** e **Desenvolvimento Dirigido por Testes**

forem levantados deverão ser discutidos posteriormente apenas pelos interessados nos problemas específicos.

- **Retrospectivas**: prática originada na Família Crystal [33] e também conhecida como *Reflection Workshops*. Elas são reuniões realizadas ao final de cada iteração na qual o processo de desenvolvimento é avaliado, a equipe discute as lições aprendidas com a experiência e planeja as mudanças para o próximo ciclo de desenvolvimento [69]. Existem diversos formatos para as reuniões de retrospectiva, mas no mais comum a equipe discute "O que funcionou bem?", "O que pode melhorar?" e "Quais problemas nos preocupam?". Normalmente, ao final da reunião, o time terá um conjunto de ações em cada uma das categorias acima e poderá priorizá-las e escolher as mais importantes para implementar na próxima iteração. As ações escolhidas devem ser capturadas em um pôster, que será anexado à Área de Trabalho Informativa. Para garantir que as idéias sejam discutidas abertamente, Kerth descreve a diretiva principal das retrospectivas [69]:

  > *"Não importa o que for descoberto, nós entendemos e acreditamos que todos fizeram o melhor trabalho possível, dado o conhecimento na época, as habilidades e os recursos disponíveis na situação em questão."*

2.9.8. Papéis na Equipe de XP

A prática do Time Completo sugere que uma equipe de XP seja formada por uma variedade de pessoas, com todas as características e habilidades necessárias para o sucesso do projeto. A primeira edição de XP estava muito mais voltada para programadores [9], porém na segunda edição Kent Beck descreve a importância da valorização de todos os outros papéis dentro de uma equipe [12]. Vale ressaltar que os papéis podem ser assumidos por pessoas diferentes, em momentos distintos e que uma mesma pessoa pode desempenhar mais de um papel. A idéia é proporcionar um ambiente produtivo no qual cada membro possa contribuir da melhor forma para o projeto. Alguns dos papéis que podem fazer parte de uma equipe de XP são:

- **Programadores**: Responsáveis por estimar histórias e tarefas, quebrar histórias em tarefas, escrever testes e código, automatizar processos tediosos e melhorar o design do sistema. Existem dois papéis especiais para programadores. Geralmente, o mais experiente em XP atua como *coach* (treinador), verificando e auxiliando os membros na execução das práticas no dia-a-dia. Já o *tracker* está constantemente coletando e compartilhando dados sobre o andamento do projeto e do processo. O *tracker* é responsável por criar e espalhar cartazes e gráficos na Área de Trabalho Informativa.

- **Arquitetos**: Procuram e executam refatorações de larga escala no sistema, escrevem testes de carga automatizados para definir cenários de estresse e auxiliam os programadores no particionamento do sistema, mantendo a ênfase no design de alto nível.

- **Analistas de Teste**: Trabalham com o cliente e com os analistas de negócio para escrever testes de aceitação automatizados, definindo os cenários de sucesso e erro de cada história. Além disso, também treinam os programadores em técnicas e ferramentas de teste.

- **Analistas de Negócio**: Trabalham com o cliente para definir as histórias do sistema e auxiliam os programadores a interpretar o valor de negócio de cada funcionalidade.

- **Projetistas de Interação**: Avaliam o modo como o sistema está sendo utilizado pelos usuários finais, identificando e sugerindo novas histórias e melhorias na interface gráfica.

- **Gerentes de Projeto**: Facilitam a comunicação dentro do time, removendo empecilhos e coordenando a comunicação com pessoas externas à equipe do projeto (fornecedores, clientes externos ou o resto da organização).

- **Gerentes de Produto**: Escrevem e priorizam histórias para o Ciclo Semanal e definem os temas para o Ciclo Trimestral. Além disso, encorajam a comunicação entre a equipe e o cliente para garantir que as preocupações e necessidades mais imediatas do cliente e do usuário final sejam atendidas.

- **Executivos**: Trazem confiança, coragem e responsabilidade para a equipe. Além disso, avaliam os objetivos do time em relação às metas da organização, monitorando e facilitando a criação de um ambiente voltado à melhoria contínua.

- **Redatores Técnicos**: Por olharem o sistema do ponto de vista do usuário final, os redatores técnicos trazem *feedback* rápido sobre as funcionalidade do sistema e estreitam o relacionamento da equipe com o cliente, levantando dúvidas e sugerindo melhorias.

- **Usuários**: Por utilizar o sistema diariamente, podem ajudar a escrever e escolher histórias e tomar decisões de domínio durante o desenvolvimento. Por representar toda a comunidade de usuários, é interessante que tenham experiência com sistemas similares para tomar as decisões mais adequadas.

- **Recursos Humanos**: Cuidam de problemas burocráticos como contratação e avaliações. Kent Beck sugere a avaliação do time ao invés de avaliações individuais para evitar conflitos internos que atrapalhem o bom andamento do projeto [12].

2.9.9. Formas de Adoção e Conclusões

Este capítulo apresentou um dos Métodos Ágeis mais conhecidos, a Programação Extrema, descrevendo seus valores, princípios e práticas. A escolha da melhor forma de adotar XP deve levar em conta todos os fatores discutidos neste capítulo e a abordagem de implantação pode variar de equipe para equipe. Enquanto algumas se sentem confortáveis com a abordagem mais rígida proposta na primeira edição, aplicando todas as 12 práticas de uma vez, outras podem preferir começar de forma mais gradual, com algumas das práticas primárias antes de partir para as práticas corolárias.

Kent Beck discute sobre os diferentes estilos de adoção fazendo uma analogia com as formas de se entrar numa piscina [11]: alguns preferem entrar de forma cuidadosa e gradual, "um pé de cada vez", evitando grandes estragos porém gastando mais tempo; outros preferem entrar de

uma vez, no estilo "bola de canhão", espalhando bastante água e passando por uma fase inicial caótica, que pode trazer maiores benefícios no curto prazo; por fim, equipes que precisam de um resultado rápido sem o risco da fase caótica, podem adotar o estilo "mergulho de cabeça" com a ajuda de um treinador externo, que vai tornar a entrada na água mais suave, pela experiência adquirida em outras situações.

Kent Beck ainda sugere alguns pontos de atenção que devem ser discutidos pela equipe para escolher a forma de adoção mais apropriada:

- Em quanto tempo o time precisa dos resultados?
- O quão dramático devem ser os resultados?
- Quanto a organização está disposta a gastar em ajuda externa?
- O quão forte são as relações entre os membros da equipe e entre a equipe e o resto da organização?

XP não deve ser utilizado em organizações cujos valores reais não se alinham com os valores de XP. Organizações que preferem dar valor a segredos, isolamento, complexidade, timidez e desrespeito não terão sucesso com a adoção de XP. Vale ressaltar ainda que uma adoção de sucesso de XP precisa abraçar os valores e princípios por trás das práticas. A adoção de algumas práticas pode trazer um pequeno benefício no curto prazo, mas as melhorias mais amplas propostas por XP só serão atingidas se houver sinergia entre os valores da equipe e de XP. Um grande choque cultural pode prejudicar a adoção de XP [26].

O discurso de XP mudou desde seu lançamento em 1999: enquanto a primeira edição enfatizava mais "como" XP funciona, a segunda edição enfatiza muito mais o "por quê". Enquanto a primeira edição era mais voltada para os programadores, a segunda edição tem um discurso mais inclusivo e flexível, trazendo benefícios para todos os envolvidos no desenvolvimento de software.

3. Métricas de Acompanhamento

Conforme apresentado no Capítulo 2, os Métodos Ágeis promovem um processo empírico para o desenvolvimento de software. Essa abordagem exige um ciclo constante de inspeção, adaptação e melhoria. Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples. Isso leva a uma proliferação de medidas baseadas na premissa de que se cada parte do processo for otimizada, os resultados do processo como um todo serão otimizados também. No entanto, essa premissa não se mostra verdadeira. Ao tentar micro-otimizar partes de um sistema por meio de diversas métricas, o verdadeiro objetivo se perde em meio a tantos substitutos e a equipe perde sua capacidade de tomar decisões de compromisso (*trade-off*) [86]. Além disso, a preocupação com as medidas erradas pode gerar incentivos errados, levando a conseqüências indesejáveis. Goldratt diz que as pessoas se comportam de acordo com a forma com que estão sendo medidas: *"Diga-me como serei avaliado e eu lhe direi como me comportarei"* [50].

Escolher as melhores formas de medição é uma tarefa do *tracker* numa equipe de XP. Jeffries [61] e Auer [3] descrevem o papel do *tracker* como alguém responsável por prover informações para a equipe sobre o progresso do time, utilizando as métricas apropriadas para destacar os pontos de melhoria e atualizando regularmente essas informações nos gráficos e pôsteres na Área de Trabalho Informativa, que Cockburn chama de *radiadores de informação* [34].

Este capítulo apresenta os conceitos relacionados às métricas para auxiliar o *tracker* de uma equipe ágil, discutindo definições, classificações, diferentes abordagens para escolha das melhores métricas e, por fim, apresentando alguns exemplos que podem ser utilizados no

acompanhamento de uma equipe ágil.

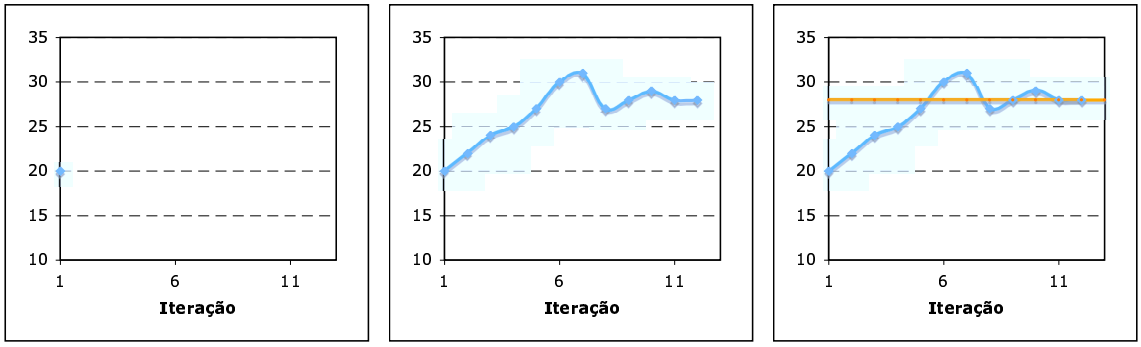
3.1. Definições

Para discutir o papel das métricas no acompanhamento de projetos ágeis, primeiro é preciso definir alguns conceitos que geralmente são usados de forma descuidada para representar a mesma coisa. Em particular, é importante conhecer as diferenças sutis entre os conceitos de ***medidas***, ***métricas*** e ***indicadores***.

Segundo o IEEE, uma ***medida*** é uma avaliação em relação a um padrão [58]; McGarry diz que é a avaliação de um atributo segundo um método de medição específico, funcionalmente independente de todas as outras medidas e capturando informação sobre um único atributo [77]. Um exemplo de medida é *5cm*: centímetro é o padrão e 5 é a medida, que indica quantos múltiplos ou frações do padrão estão sendo representados. Em desenvolvimento de software, um exemplo de medida pode ser o número de linhas de código. No entanto, não existe um padrão universal para representar linhas de código, pois as linguagens podem variar, assim como as regras para cálculo de linhas de código. Portanto, uma medida pode ser baseada em um padrão local ou universal, mas o padrão precisa ser bem definido.

Uma ***métrica*** é um método para determinar se um sistema, componente ou processo possui um certo atributo [57]. Ela é geralmente calculada ou composta por duas ou mais medidas. Um exemplo de métrica é o *número de defeitos encontrados após a implantação*: as medidas que compõem essa métrica são o *número de defeitos* e a *fase (ou data) onde o defeito foi identificado*.

Um ***indicador*** é um aparelho ou variável que pode ser configurado para um determinado estado com base no resultado de um processo ou ocorrência de uma determinada condição. Por exemplo: um semáforo ou uma *flag* [57]. Conforme a definição do IEEE, um indicador é algo que chama a atenção de uma pessoa para uma situação particular. Ele geralmente está relacionado a uma métrica e provê a interpretação daquela métrica numa determinada situação



(a) Medida (b) Métrica (c) Indicador

Figura 3.1.: Total de Pontos Entregues por Iteração

ou contexto. Por exemplo: um aumento substancial no número de defeitos encontrados na última versão pode ser um indicador de que a qualidade do software piorou.

O seguinte exemplo fictício demonstra a relação entre medidas, métricas e indicadores: as estimativas das histórias num projeto em XP são geralmente medidas em pontos. Pontos representam o tamanho e a complexidade de uma história em relação a outras histórias do mesmo projeto. Equipes ágeis separam a estimativa do tamanho de uma história do tempo que ela demora para ser implementada. O sistema de pontos se baseia em uma escala numérica definida pela equipe que permite a estimativa por comparação. Uma história de 4 pontos tem mais ou menos o dobro do tamanho de uma história de 2 pontos. Algumas escalas comumente utilizadas são escalas exponenciais $(1, 2, 4, 8, 16, \ldots)$ ou uma seqüência de Fibonacci $(1, 2, 3, 5, 8, \ldots)$. Segundo Cohn, essas escalas funcionam bem pois os valores mais altos incluem maior incerteza, refletindo a natureza preditiva das estimativas [35]. Supondo que, ao término da primeira iteração de um projeto, constata-se que a equipe entregou 4 histórias, somando um total de 20 pontos (Figura 3.1(a)). Conforme a equipe vai terminando as próximas iterações, percebe-se que o número total de pontos entregues aumenta aos poucos. Após um certo tempo, essa tendência de subida é interrompida e o número total de pontos entregues cai um pouco, atingindo um patamar (Figura 3.1(b)).

A Figura 3.1(a) representa uma medida. Sem nenhuma outra informação para comparar ou uma tendência para seguir, uma medida não provê muita informação. A Figura 3.1(b) representa uma métrica, nesse caso a velocidade da equipe. Uma métrica é composta por diversas medidas como o número de pontos entregues e o número da iteração terminada. Por fim, a Figura 3.1(c) representa um indicador. Ele dá um contexto para a métrica, servindo como base para comparação. Uma métrica é sempre interpretada sob um ponto de vista específico. Por isso, é possível derivar diversos indicadores a partir da mesma métrica. O significado de um indicador sempre depende de um contexto, portanto duas equipes podem analisar a mesma métrica de forma diferente. Supondo que essa mesma equipe, em outro projeto, tenha obtido uma velocidade inferior. Ao analisar essa mesma métrica (Figura 3.1(b)), tal equipe consideraria uma melhora na velocidade da equipe, enquanto no exemplo citado, ao invés de uma melhora, houve apenas uma estabilização.

A palavra métrica será utilizada neste texto daqui em diante. No entanto, sempre que uma métrica for avaliada ou analisada, o conceito de um indicador estará sempre implícito. Da mesma forma, toda métrica depende de medidas, portanto elas também estão sendo consideradas.

3.2. Classificações

As métricas podem ser classificadas segundo diferentes critérios. Esta seção apresenta algumas das possíveis classificações que um *tracker* precisa considerar quando utilizar uma métrica.

3.2.1. Objetiva/Subjetiva

Conforme discutido anteriormente, uma métrica é composta por medidas que avaliam atributos de um objeto. O valor de uma métrica ***objetiva*** depende somente do objeto em questão e não do ponto de vista de quem a está interpretando. Por exemplo: número de *commits* no repositório é uma métrica objetiva, pois é obtida diretamente da ferramenta. Por outro lado, o

valor de uma métrica ***subjetiva*** depende do objeto em questão e também do ponto de vista de quem a está interpretando. Um exemplo de métrica subjetiva é a qualidade do código, numa escala de 0% a 100%. Apesar da escala definir um intervalo numérico, a natureza da métrica ainda é subjetiva, pois depende do ponto de vista de quem está avaliando. Os critérios para definir a "qualidade do código" podem variar de pessoa para pessoa. Métricas objetivas são mais fáceis de serem obtidas de forma automatizada.

3.2.2. Quantitativa/Qualitativa

Além da natureza objetiva/subjetiva, uma métrica pode ainda ser classificada como quantitativa ou qualitativa. O valor de uma métrica ***quantitativa*** pertence a um intervalo de uma certa magnitude e geralmente é representado por um número. Tal estrutura permite que medidas quantitativas sejam comparadas entre si. A maioria dos exemplo utilizados até aqui neste capítulo representam métricas quantitativas, como o número de linhas de código ou o número de defeitos encontrados. Por outro lado, valores de uma métrica ***qualitativa*** são aqueles representados por palavras, símbolos ou figuras ao invés de números [48]. Um exemplo de métrica qualitativa é o humor da equipe ou a satisfação do cliente.

A maioria dos estudos empíricos em engenharia de software usam uma combinação entre métodos quantitativos e qualitativos. Uma das formas mais comuns de combinar ambas as estratégias é a ***codificação***, que consiste na extração de valores quantitativos dos dados qualitativos para permitir o tratamento estatístico ou outra abordagem quantitativa [99]. Vale ressaltar que a classificação de uma métrica como quantitativa ou qualitativa é ortogonal à classificação como objetiva ou subjetiva. Geralmente uma métrica quantitativa é objetiva e uma qualitativa é subjetiva, mas isso não é sempre verdade. O processo de codificação transforma uma métrica qualitativa em quantitativa, mas não altera sua objetividade ou subjetividade. Por exemplo, considere a seguinte frase que constitui um fragmento de dado qualitativo: "João, Maria e Pedro foram os únicos que participaram da reunião". Isso poderia ser transformado

num dado quantitativo como: "numero_de_participantes = 3". A informação continua objetiva após a codificação. Além disso, uma parte da informação é perdida, pois não sabemos mais os nomes dos participantes. Considerando agora outro exemplo de dado qualitativo: "Paulo disse que essa classe Java é bem simples de entender e sua complexidade é bem menor que as outras classes do sistema". Tal informação poderia ser codificada para o seguinte dado quantitativo: "complexidade = baixa". Houve novamente perda de informação no processo de codificação e, apesar do valor quantitativo parecer mais objetivo, continua tão subjetivo quanto antes.

3.2.3. Organizacional/Acompanhamento

Hartmann e Dymond sugerem uma outra categoria para classificação, distinguindo métricas organizacionais de métricas de diagnóstico [53]. Neste texto, ao invés de usar o termo "diagnóstico" usarei o termo "acompanhamento" por estar mais alinhado com o tema do trabalho e por compartilhar as mesmas características de "diagnóstico" propostas por Hartmann e Dymond.

Métricas organizacionais são aquelas que medem a quantidade de valor de negócio entregue ao cliente. Essa definição levanta algumas perguntas: em primeiro lugar, quem é o cliente? Collins sugere que, no nível organizacional, o cliente deve ser o dono ou o responsável pelo produto (*stakeholder*) ou talvez o usuário final [36]. Isso deixa a segunda pergunta em aberto, o que é valor? No seu livro, Collins discute os atributos e comportamentos em comum das empresas que passaram de um longo histórico de resultados medíocres para um longo histórico de resultados extraordinários. Ele mostra que as empresas que foram do bom para o ótimo escolheram um fator-chave único de direcionamento econômico como métrica para auxiliar na tomada de decisão. Idealmente essa métrica-chave deve ser definida pelos executivos da empresa, porém em projetos de código aberto, onde o objetivo final nem sempre é financeiro, outros fatores de sucesso como número de usuários ou a satisfação do usuário podem ser utilizados. A seção 3.5.2 apresentará com mais detalhes alguns exemplos de métricas

organizacionais.

Métricas de acompanhamento provêem informações que ajudam o time no entendimento e melhoria do processo que produz valor de negócio. Uma vez que uma métrica organizacional ampla é definida, a equipe precisa de medições locais para auxiliá-los a atingir o objetivo. Essas métricas agregam dados porém não os associam a nenhum indivíduo. Elas existem e têm validade dentro de um contexto particular, porém recomenda-se que elas sejam escolhidas com cuidado e utilizadas somente durante um certo período de tempo. Métricas de acompanhamento devem ser descartadas uma vez que tenham servido seu propósito. A meta mais ampla definida pela métrica organizacional deve guiar a utilização de diferentes métricas de acompanhamento temporárias. Um exemplo de métrica de acompanhamento já citado neste capítulo é a velocidade da equipe.

Poppendieck utiliza uma nomenclatura diferente para as métricas organizacionais e de acompanhamento, chamando-as de métricas de avaliação de desempenho e métricas informativas, respectivamente [85]. Apesar dos nomes serem diferentes, as características descritas nesta seção são as mesmas.

3.3. O Que Medir?

Esta seção apresenta algumas abordagens para escolher quais métricas utilizar, apresentando também algumas das características de uma boa métrica ágil. Um *tracker* deve utilizar as abordagens apresentadas aqui juntamente com o conhecimento sobre sua equipe ao escolher as melhores métricas para sua situação.

3.3.1. Abordagem Objetivo-Pergunta-Métrica (Goal Question Metric)

Uma das abordagens mais conhecidas e utilizadas em estudos empíricos em engenharia de software é a abordagem objetivo-pergunta-métrica (*Goal Question Metric* ou GQM), proposta por Basili [6]. O modelo de medição proposto pelo GQM é composto de três níveis:

- **Nível Conceitual (Objetivo)**: Um objetivo é definido para um objeto em relação a algum modelo de qualidade, a partir de diversos pontos de vista e para um ambiente específico. Um objeto pode ser um produto (documento, código-fonte, testes, etc.), um processo (especificação, entrevista, codificação, etc.) ou um recurso (pessoas, hardware, espaço físico, etc.).

- **Nível Operacional (Pergunta)**: Um conjunto de perguntas caracterizam a forma de avaliação e cumprimento do objetivo escolhido. As perguntas tentam relacionar o objeto de estudo com as características de qualidade desejáveis a partir do ponto de vista definido.

- **Nível Quantitativo (Métrica)**: Um conjunto de dados é associado com cada pergunta para tentar encontrar uma forma quantitativa de respondê-la. Métricas podem ser objetivas ou subjetivas.

Esse modelo define uma estrutura hierárquica que começa com um objetivo claro. O formato proposto pela abordagem GQM para definir um objetivo é composto de: uma motivação, uma preocupação ou tópico, um objeto e um ponto de vista. A partir desse objetivo, uma série de perguntas são definidas e, a partir dessas perguntas, uma série de métricas são escolhidas. A mesma métrica pode ser usada para responder diferentes perguntas. Da mesma forma, métricas e perguntas podem ser utilizadas em mais de um modelo GQM para diferentes objetivos, porém a forma de medição deve levar em conta os diferentes pontos de vista de cada modelo. A Tabela 3.1 apresenta um exemplo de um modelo GQM.

Segundo Basili, a definição de um objetivo deve se basear em três fontes de informação [6]: a primeira fonte de informação são as políticas e estratégias organizacionais, de onde derivam a motivação e o tópico do objetivo; a segunda fonte são as descrições dos processos e produtos da empresa, de onde deriva o objeto de avaliação; por fim, a terceira fonte de informação é o modelo organizacional da empresa, de onde derivam os pontos de vista que serão levados em conta na avaliação do objetivo.

Objetivo	Motivação Tópico Objeto Ponto de Vista	Melhorar o tempo gasto no processo de correção de defeitos sob o ponto de vista gerencial.
Pergunta		Qual é a velocidade atual de correção de um defeito?
Métricas		Tempo médio de ciclo Desvio padrão % de casos acima do limite máximo
Pergunta		O processo de correção de defeitos está melhorando?
Métricas		$\frac{\text{Tempo médio de ciclo atual}}{\text{Tempo médio de ciclo desejado}} * 100$ Avaliação subjetiva do gerente responsável

Tabela 3.1.: Exemplo de modelo Objetivo-Pergunta-Métrica

3.3.2. Abordagem Lean

Ao adaptar os conceitos Lean que funcionaram bem para a manufatura, cadeia de suprimentos e desenvolvimento de produtos, Poppendieck propõe uma abordagem para escolha das métricas mais apropriadas [85, 86]. Conforme discutido na Seção 3.2.3, a abordagem Lean distingue bem as métricas organizacionais das métricas de acompanhamento e um de seus princípios para o desenvolvimento de software é "Otimizar o todo". Sua proposta para o uso de métricas na avaliação de desempenho é medir sempre um nível acima.

Uma tendência natural para avaliação de desempenho é a decomposição. O senso comum diz que se as partes de um sistema forem otimizadas, o sistema todo também o será. No entanto a micro-otimização tende a degradar o resultado geral pois não é possível medir tudo. Austin discute os problemas da avaliação de desempenho e destaca que sua principal vantagem (*"Você tem o que você mede"*) é também seu principal problema (*"Você tem exatamente o que você mede, e nada mais"*). Fatores importantes acabam ficando fora da decomposição por não serem facilmente mensurados, como: criatividade, *insight*, colaboração, dedicação e preocupação com a satisfação do cliente [4].

Quando a avaliação de um indivíduo é realizada por uma métrica que leva em conta apenas

fatores dentro do seu campo de influência, ele tende a trabalhar para otimizar essa métrica, deixando de pensar no sistema como um todo. A prática sugerida pela abordagem Lean é medir sempre em um nível acima. Quando a avaliação leva em conta o time todo ou a empresa toda, ela gera incentivos para que os indivíduos trabalhem de forma colaborativa para atingir um resultado comum. Por isso, ao invés de criar diversas métricas, é mais importante reduzir o número de métricas organizacionais, escolhendo uma que defina um objetivo amplo, gerando os incentivos que farão com que o comportamento dos sub-sistemas otimizem o todo [86].

Além das métricas organizacionais, a abordagem Lean também sugere o uso de métricas de acompanhamento para auxiliar a equipe. No entanto, essas métricas devem ser definidas de forma a ocultar o desempenho individual. Quando a contagem de defeitos leva em conta o indivíduo que causou o erro, ela passa a ser uma métrica de avaliação, gerando os incentivos errados. Deming diz que a baixa qualidade nunca é responsabilidade de um indivíduo, mas sim de um processo de gerenciamento que permite que a ausência de defeitos seja impossível [40]. Por isso, é importante que um métrica de acompanhamento seja totalmente desassociada de qualquer avaliação de desempenho. Seu propósito deve ser meramente informacional.

3.3.3. Retrospectivas

Reuniões de Retrospectiva, descritas na Seção 2.9.7, encorajam a discussão constante sobre o processo e a forma de trabalho da equipe. Elas são o momento de reflexão que deve fazer parte do ciclo de inspeção e adaptação proposto pelos Métodos Ágeis. Segundo Cockburn, o criador da Família Crystal onde a Retrospectiva é prática fundamental, elas ajudam a encontrar o processo mais aceitável para a equipe [34].

O resultado de uma reunião de Retrospectiva costuma ser um pôster destacando os principais pontos de melhoria que a equipe escolheu se concentrar na próxima iteração. A partir dessa discussão, o *tracker* pode escolher algumas métricas de acompanhamento para auxiliar a equipe a entender o progresso em relação aos pontos de melhoria levantados. A Seção 3.5.3 e o

Apêndice B apresentam exemplos de métricas de acompanhamento para auxiliar o *tracker* e a equipe nessa escolha.

3.3.4. Características de Uma Boa Métrica Ágil

Com base em diversas fontes, Hartman e Dymond propõe uma compilação de algumas das heurísticas que um *tracker* deve considerar quando estiver escolhendo uma métrica para sua equipe [53]. Uma boa métrica ágil deve:

- **Reforçar princípios ágeis**: colaboração com o cliente e entrega de valor são princípios fundamentais para os Métodos Ágeis.
- **Medir resultados e não saídas**: ao valorizar a simplicidade, os melhores resultados podem ser aqueles que minimizam a quantidade de trabalho realizado (saídas).
- **Seguir tendências e não números**: os valores representados por uma métrica são menos importantes que a tendência demonstrada. Ao medir a velocidade da equipe, é melhor se preocupar com sua estabilização do que com o valor absoluto que ela representa.
- **Responder uma pergunta específica para uma pessoa real**: toda métrica deve expor informação para um ponto de vista específico. Se outra pessoa tem outra pergunta, é melhor usar outra métrica.
- **Pertencer a um conjunto pequeno de métricas e diagnósticos**: é impossível medir tudo. Muita informação pode esconder o que realmente importa. Minimize o número de métricas e meça o que é mais importante.
- **Ser facilmente coletada**: para métricas de acompanhamento objetivas e quantitativas, o ideal é ter uma coleta automatizada.
- **Revelar, ao invés de esconder, seu contexto e suas variáveis**: uma métrica deve deixar claro os fatores que a influenciam para evitar manipulações e facilitar a melhoria

do processo.

- **Incentivar a comunicação**: uma métrica isolada de seu contexto perde o sentido. Um bom sinal é quando as pessoas comentam o que aprenderam com uma métrica.

- **Fornecer *feedback* freqüente e regular**: para amplificar o aprendizado e acelerar o processo de melhoria, as métricas devem ser freqüentemente atualizadas e disponibilizadas na Área de Trabalho Informativa.

- **Encorajar um alto nível de qualidade**: o nível aceitável de qualidade deve ser definido pelo cliente e não pela equipe. Os Métodos Ágeis exigem sempre um alto nível de qualidade do software desenvolvido.

3.4. Discussão

As abordagens apresentadas na Seção 3.3 possuem vantagens e desvantagens e o *tracker* de uma equipe ágil deve saber balanceá-las. O modelo GQM propõe uma abordagem *top-down* para a definição de métricas que por um lado esclarece os objetivos por trás de cada métrica e evita a medição das coisas erradas, porém por outro lado sua estrutura hierárquica estimula a proliferação de diversas métricas. Já a abordagem Lean distingue bem os dois níveis de medição (organizacional/acompanhamento) e estimula um menor número de métricas organizacionais. A prática de medir sempre um nível acima evita a proliferação de métricas e cria os incentivos para a colaboração entre todos os responsáveis pelo fluxo de entrega de valor do sistema. Apesar de ambas as abordagens funcionarem para a definição e escolha das métricas organizacionais, não dão muita direção em relação a escolha das métricas de acompanhamento. As reuniões de Retrospectiva resolvem esse problema, fazendo com que a própria equipe discuta e reflita sobre os pontos do processo que podem ser melhorados. Essa discussão auxilia a escolha das melhores métricas de acompanhamento.

Com base nas características apresentadas na Seção 3.3.4, Hartman e Dymond sugerem

que o *tracker* utilize uma lista de verificação ao escolher uma métrica [53]. Levando em consideração a discussão sobre as diferentes abordagens aqui apresentada, o autor propõe uma lista adaptada:

- **Nome**: deve ser bem escolhido para evitar confusão e ambigüidade.
- **Classificação**: conforme apresentado na Seção 3.2.
- **Objetivo**: conforme definido no modelo GQM, incluindo uma motivação, uma preocupação ou tópico, um objeto e um ponto de vista.
- **Pergunta**: conforme definido no modelo GQM, toda métrica deve responder uma pergunta específica.
- **Base de Medição**: uma clara definição das medidas utilizadas para cálculo da métrica.
- **Suposições**: devem ser identificadas para um claro entendimento do que os dados estão representando.
- **Tendência Esperada**: uma idéia de qual o comportamento esperado para a métrica.
- **Quando utilizar?**: deve esclarecer os motivos que levaram à criação da métrica e, caso a métrica já tenha sido utilizada anteriormente, mostrar um pouco do seu histórico.
- **Quando parar de utilizar?**: é importante saber até quando uma métrica será útil, antes mesmo de utilizá-la, principalmente para métricas de acompanhamento.
- **Formas de Manipulação**: deve esclarecer como as pessoas tentarão alterar seu comportamento em função da métrica para gerar números "mais favoráveis".
- **Cuidados e Observações**: recomendações sobre outras métricas similares, limites no uso e perigos associados à má utilização da métrica.

3.5. Exemplos

Esta seção apresenta alguns exemplos de medidas e métricas organizacionais e de acompanhamento. Algumas dessas medidas e métricas serão utilizadas no estudo de caso do Capítulo 4. Essa lista não pretende ser exaustiva, enumerando todas as possíveis medidas e métricas. Mais exemplos, incluindo algumas situações de aplicação num projeto XP, podem ser encontrados no Apêndice B.

3.5.1. Medidas

As medidas apresentadas nesta seção serão separadas de acordo com sua respectiva classificação (conforme descrito na Seção 3.2). Abaixo seguem exemplos de medidas quantitativas e objetivas. Por sua natureza, elas podem ser coletadas de forma automatizada, conforme será descrito na Seção 4.1.

- **Total de Linhas de Código (TLOC)**: representa o número total de linhas de código de produção do sistema, descartando linhas em branco e comentários. Outra forma comum de utilização é contar aos milhares (*Thousand Lines of Executable Code* ou KLOEC[1]).

- **Total de Linhas de Código de Teste (TLOTC)**: representa o número total de pontos de teste do sistema, conforme definido por Dubinsky [41]. Um ponto de teste é considerado como um passo do cenário de um teste de aceitação automatizado ou como uma linha de código de teste unitário automatizado, descartando linhas em branco e comentários.

- **Número de Linhas Alteradas**: representa o número de linhas (não apenas código-fonte) adicionadas, removidas e atualizadas no repositório de código.

- **Número de *Commits*** representa o número de *commits* efetuados no repositório de código.

[1] o "E" de *Executable* reforça a idéia de que linhas com comentário ou espaços em branco não são consideradas

- **Estimativas Originais**: representa o total de pontos (ou horas) originalmente estimadas pela equipe para todas as histórias da iteração. Beck e Fowler sugerem a utilização de "*horas ideais*" nas estimativas e controle da iteração, mas a unidade de medida efetivamente utilizada não importa tanto, contanto que seja usada consistentemente durante o projeto [16].

- **Estimativas Finais**: representa o total de pontos (ou horas, ou "horas ideais") efetivamente reportadas como gastas para implementar as histórias da iteração.

- **Número de Histórias Entregues**: representa o número total de histórias implementadas e aceitas pelo cliente.

- **Número de Pontos Entregues**: representa o número total de pontos implementados e aceitos pelo cliente.

- **Tempo**: utilizado no cálculo de diversas métricas, pode ser utilizado como duração (em meses, semanas, dias, etc.) ou como número de iterações.

- **Tamanho da Equipe**: geralmente utilizado no cálculo de métricas, pode ser utilizado como a quantidade de pessoas na equipe ou como esforço medido em pessoas-mês ou pessoas-ano.

- **Complexidade Ciclomática de McCabe ($v(G)$)**: mede a quantidade de lógica de decisão num único módulo de software [76]. Um *Grafo de controle de fluxo* descreve a estrutura lógica de um algoritmo através de vértices e arestas. Os vértices representam expressões ou instruções computacionais (como atribuições, laços ou condicionais), enquanto as arestas representam a transferência do controle entre os vértices [109]. A Complexidade Ciclomática é definida para cada módulo (num sistema orientado a objetos, cada método de uma classe representa um módulo) como $e - n + 2$, onde e e n são o número de arestas e vértices do grafo de controle de fluxo, respectivamente.

- **Métodos Ponderados por Classe** (*Weighted Methods per Class* ou WMC): mede a complexidade de uma classe num sistema orientado a objetos. Definida pela soma ponderada de todos os métodos de uma classe [30]. É comum utilizar $v(G)$ como fator de peso, então WMC pode ser calculada como $\sum c_i$, onde c_i é a Complexidade Ciclomática do i-ésimo método da classe.

- **Falta de Coesão dos Métodos** (*Lack of Cohesion of Methods* ou LCOM): mede a coesão de uma classe e é calculada através do método de Henderson-Sellers [54]. Se $m(P)$ é o número de métodos que acessam a propriedade P, LCOM é calculada como a média de $m(P)$ para todas as propriedades, subtraindo o número de métodos m e dividindo o resultado por $(1 - m)$. Um valor baixo indica uma classe coesa, enquanto um valor próximo de 1 indica falta de coesão.

- **Profundidade da Árvore de Herança** (*Depth of Inheritance Tree* ou DIT): o comprimento do maior caminho a partir de uma classe até a classe-base da hierarquia (excluindo a classe-base `Object`).

- **Número de Filhos** (*Number of Children* ou NOC): o número total de filhos imediatos de uma classe.

- **Acoplamento Aferente** (*Afferent Coupling* ou AC): o número total de classes de fora de um pacote que dependem de classes de dentro do pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-in* da classe.

- **Acoplamento Eferente** (*Efferent Coupling* ou EC): o número total de classes de dentro de um pacote que dependem de classes de fora do pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-out* da classe, ou como CBO (*Coupling Between Objects* ou Acoplamento entre Objetos) na família de métricas CK (Chidamber-Kemerer) [30].

Alguns dos fatores de desenvolvimento propostos por Boehm e Turner [24] e apresentados com mais detalhe na Seção 4.2.1 servem como exemplos de medidas quantitativas e subjetivas:

- **Dinamismo**: O quantidade de mudanças de requisitos por mês.
- **Cultura**: Uma medida da porcentagem da equipe que prefere trabalhar em um cenário caótico ao invés de um cenário ordenado, ou seja, a porcentagem da equipe capaz de aceitar mudanças durante o projeto.
- **Criticalidade**: O impacto causado por uma falha no software, podendo afetar desde uma quantia aceitável de investimento até a perda de uma vida.
- **Nível Pessoal**: A porcentagem da equipe que pertence aos diversos níveis propostos por Cockburn [34], descritos em detalhe na Seção 4.3.3.
- **Aderência às Práticas de XP**: uma forma de medir o grau de utilização de cada prática de XP. Cada integrante dá uma nota para o nível "atual" e "desejado" da equipe em relação a cada prática de XP. O questionário utilizado para coletar essa medida é descrito em mais detalhes na Seção 4.1 e está disponível no Apêndice A.

Por fim, alguns exemplos de medidas qualitativas (que podem ser codificadas para medidas quantitativas, conforme descrito na Seção 3.2.2) e subjetivas são:

- **Moral da Equipe**: mede o humor e a motivação de cada membro da equipe. Uma forma para coletar essa medida é descrita em mais detalhes na Seção 4.1.
- **Satisfação do Cliente**: mede o nível de satisfação do cliente com o produto desenvolvido.

3.5.2. Métricas Organizacionais

Esta seção apresenta quatro exemplos de métricas organizacionais, que podem ser utilizadas para avaliação de uma equipe ágil: "Funcionalidades Testadas e Entregues" é uma métrica

proposta por Ron Jeffries [60], "Tempo Médio de Ciclo" foi proposta pela abordagem Lean [85, 86], "Retorno Financeiro" é uma métrica mais amplamente citada [12,86,53] e "*Net Promoter Score*" foi proposta por Reichheld [89]. Elas serão apresentadas conforme o formato proposto na Seção 3.4.

Funcionalidades Testadas e Entregues (Running Tested Features ou RTF)

- **Classificação**: Quantitativa e subjetiva, pois o cliente define quando uma funcionalidade está pronta através dos testes de aceitação.
- **Objetivo**: Maximizar a quantidade de valor de negócio entregue em cada funcionalidades do sistema, sob o ponto de vista do cliente.
- **Pergunta**: Qual é a taxa de valor de negócio entregue por funcionalidade testada e implantada? Quando o software deixa de ser inventário e passa a agregar valor?
- **Base de Medição**: Requisitos são quebrados em histórias (Funcionalidades). Testes de aceitação automatizados são definidos pelo cliente e servem como critério para avaliar quando a história está pronta (Testada). Toda funcionalidade pronta deve estar integrada num único produto, com a possibilidade de ser implantado e prontamente utilizado (Entregue).
- **Suposições**: O cliente deve trabalhar em colaboração com a equipe, definindo histórias e cenários de aceitação. Uma funcionalidade testada e pronta só poderá trazer valor efetivo de negócio quando entrar em produção.
- **Tendência Esperada**: RTF deve crescer linearmente logo após o início do projeto e até o seu término. As funcionalidades que trazem mais valor serão implementadas primeiro.
- **Quando utilizar?**: Para avaliar a execução de projetos ágeis. Para entregar funcionalidades testadas a partir do início do projeto, a equipe vai precisar de práticas ágeis. Ela

não terá tempo de fazer muito design prematuro (*Big Design Up-Front*), assim como não poderá esquecer dos testes automatizados.

- **Quando parar de utilizar?**: Quando o projeto terminar ou o retorno financeiro não justificar seu prolongamento.

- **Formas de Manipulação**: Uma forma de manipular essa métrica é através da entrega das funcionalidades mais fáceis, ao invés das mais importantes. Definir poucos testes também fará com que a funcionalidade esteja pronta mais rapidamente.

- **Cuidados e Observações**: Medir funcionalidade entregue pode não refletir diretamente em valor de negócio. O excesso de funcionalidades é, na verdade, um dos grandes inimigos do desenvolvimento de software. É importante que as funcionalidades entregues no início do projeto sejam as mais importantes e com maior valor de negócio.

Tempo Médio de Ciclo (Cycle Time)

- **Classificação**: Qualitativa e objetiva.

- **Objetivo**: Minimizar o tempo médio de ciclo de um sistema, sob o ponto de vista do cliente final (usuário).

- **Pergunta**: Quanto tempo é gasto do conceito ao retorno financeiro (*concept to cash*)? Ou entre uma "ordem" de compra de um cliente e sua realização em forma de software entregue? Ou entre a identificação de um defeito em produção e sua resolução? O importante não é descobrir o quão rápido o sistema *pode* entregar valor, mas sim o quão rápido o sistema *entrega* valor de forma *repetitiva e confiável*.

- **Base de Medição**: Com base num mapa de fluxo de valor do sistema todo (*Value Stream Map*), que sempre e começa e termina com o cliente e, para cada parte do processo, mapeia quanto tempo é gasto agregando valor ao sistema e quando tempo é

desperdiçado em tarefas que não agregam valor [85]. O tempo de ciclo é calculado de forma objetiva através da medição do tempo que uma "ordem" leva para passar pelo mapa de fluxo de valor do sistema.

- **Suposições**: A equipe conhece as etapas do processo por onde uma "ordem" deve passar e o processo está mapeado num mapa de fluxo de valor. O tempo total gasto para processar a "ordem" considera todos os tipos de desperdício no sistema: falta de habilidades, atrasos, *hand-offs* entre diferentes equipes, etc.

- **Tendência Esperada**: O tempo médio de ciclo do sistema deve diminuir gradualmente conforme o processo é melhorado, até atingir um patamar mínimo. No entanto, a equipe tem autonomia para estar constantemente desafiando o processo e melhorando-o continuamente, diminuindo ainda mais o tempo médio de ciclo.

- **Quando utilizar?**: Para avaliar o comprometimento das pessoas com o sistema todo. Ela exige que outras métricas mais locais sejam otimizadas, porém o contrário não é verdade. A preocupação em otimizar subsistemas provavelmente fará com que o tempo de ciclo aumente [86].

- **Quando parar de utilizar?**: Quando o sistema não precisar mais produzir valor.

- **Formas de Manipulação**: Por ser uma métrica ampla, essa métrica cria os incentivos para que as equipes cruzem barreiras do processo, exigindo que as pessoas colaborem para criar um produto de qualidade. Dessa forma, fica difícil manipular essa medida.

- **Cuidados e Observações**: O tempo médio de ciclo tende a diminuir quando o sistema trabalha com peças pequenas (*small batches*). No caso do desenvolvimento de software, por exemplo, essas peças podem ser histórias ou defeitos. Se o tempo médio de ciclo se estabilizar, isso não significa que a equipe chegou ao valor ótimo. Ela deve estar constantemente desafiando os padrões atuais para encontrar formas de reduzir o tempo médio de ciclo [86].

Retorno de Investimento (Return of Investment ou ROI)

- **Classificação**: Quantitativa e objetiva.
- **Objetivo**: Minimizar o tempo gasto até um sistema trazer retorno financeiro, sob o ponto de vista do cliente.
- **Pergunta**: Qual é a taxa de retorno financeiro do projeto? Quando ele começará a ser obtido? Qual o lucro esperado com o investimento num sistema de software?
- **Base de Medição**: Através de análise do fluxo de caixa financeiro por iteração.
- **Suposições**: A entrega de valor será realizada ao final da iteração, no entanto o retorno financeiro só será obtido quando o software entrar em produção.
- **Tendência Esperada**: As funcionalidades com maior valor e maior retorno serão entregues no início do projeto e, conforme as outras funcionalidades vão sendo implementadas, o retorno obtido por funcionalidade vai reduzindo.
- **Quando utilizar?**: Ao analisar, priorizar e executar projetos onde algum retorno financeiro é esperado. Essa métrica deve ser compreendida por todos os envolvidos no projeto.
- **Quando parar de utilizar?**: Quando o investimento for retirado ou quando as funcionalidades deixarem de agregar valor financeiro justificável.
- **Formas de Manipulação**: Entregar as funcionalidades menores, ao invés das que trazem mais valor financeiro, no início do projeto para que o ROI comece a ser obtido rapidamente. Isso pode afetar a taxa de retorno financeiro esperada.
- **Cuidados e Observações**: Assim que o software entra em produção, com um subconjunto das funcionalidades esperadas, o modelo de retorno financeiro deve ser reavaliado e as funcionalidades restantes re-priorizadas para levar em conta os dados reais de

retorno financeiro. Além disso, a equipe toda deve ser exposta ao modelo de lucros e perdas que afetará o resultado financeiro do projeto, para que tenham o conhecimento necessário ao tomar decisões de compromisso (*trade-off*) durante o desenvolvimento.

Outras métricas para avaliação de retorno financeiro, como *Net Present Value* (NPV) ou *Internal Rate of Return* (IRR) também podem ser utilizados como métrica organizacional.

Net Promoter Score ou NPS

- **Classificação**: Quantitativa e subjetiva, pois depende da avaliação da satisfação do cliente.
- **Objetivo**: Maximizar a satisfação do cliente final (usuário), sob o ponto de vista do cliente (do projeto).
- **Pergunta**: Como distinguir entre um dólar de lucro do tipo bom, que trará crescimento, e um dólar de lucro do tipo ruim, que prejudicará o crescimento? Como avaliar a satisfação do cliente final de um projeto de software?
- **Base de Medição**: Calculada com base numa única e simples questão para o cliente final: "O quanto você recomendaria a empresa ou o produto X para um amigo ou colega?". Uma nota de 0 (não recomendaria) a 10 (definitivamente recomendaria) é obtida. Clientes com nota entre 9 e 10 são chamados de promotores, notas entre 7 e 8 representam clientes neutros, enquanto notas entre 0 e 6 representam clientes afastadores (*retractors*). O NPS é calculado subtraindo a porcentagem de clientes afastadores da porcentagem de clientes promotores, podendo variar entre -100% e 100%.
- **Suposições**: Seu produto atinge pessoas com poder de influência no público em geral.
- **Tendência Esperada**: Valores negativos do NPS representam um sério risco. Dessa forma, a tendência esperada para esta métrica é de crescimento. Empresas boas possuem um NPS médio de 10%, enquanto empresas realmente boas conseguem NPS de 50% [89].

- **Quando utilizar?**: Quando o produto não possuir intenções financeiras (projetos de código aberto, por exemplo) ou quando seu público alvo tiver influência no sucesso do produto.

- **Quando parar de utilizar?**: Quando o produto não estiver mais tentando penetração no mercado ou quando for finalizado.

- **Formas de Manipulação**: Como a influência da equipe nos clientes finais do produto é praticamente nenhuma, fica difícil manipulá-los para obter um NPS melhor.

- **Cuidados e Observações**: O valor bruto do NPS não pode usado para comparações diretas entre empresas ou linhas de produto devido a diferenças nos segmentos de mercado ou até a diferenças geográficas e culturais entre os clientes finais. Ele deve ser usado como um indicador de tendência dentro de uma única empresa ou produto.

3.5.3. Métricas de Acompanhamento

Esta seção apresenta três exemplos de métricas de acompanhamento, que podem ser utilizadas para auxiliar na melhoria do processo utilizado pela equipe para trazer valor de negócio. A "Velocidade" é uma métrica amplamente utilizada por equipes ágeis [12,16,98,53], enquanto o "Fator de Integração" e o "Fator de Teste" foram propostos pelo autor que, apesar de não conhecer uma citação específica na literatura, acredita que sua autoria não seja original. As métricas serão apresentadas conforme o formato proposto na Seção 3.4. Por sua natureza mais descartável, o objetivo de uma métrica de acompanhamento, no formato proposto pelo modelo GQM, deve ser sempre "Melhorar a adoção da prática X sob o ponto de vista da equipe".

Velocidade

- **Classificação**: Quantitativa e objetiva.

- **Pergunta**: Quanto software a equipe consegue entregar por iteração?

- **Base de Medição**: Pontos (*story-points*) ou "horas ideais" entregues por iteração.

- **Suposições**: A equipe está entregando software funcionando a cada iteração.

- **Tendência Esperada**: A velocidade pode ser afetada por diversos fatores, como: mudança na equipe, impedimentos, conhecimento das ferramentas e tecnologias utilizadas, etc. Um time estabilizado, durante um mesmo projeto e com todos os recursos necessários disponíveis tende a aumentar sua velocidade durante um certo tempo (geralmente no início), até atingir um patamar, onde ela se estabiliza.

- **Quando utilizar?**: A velocidade é uma métrica de acompanhamento muito útil para a equipe conhecer seu limite e sempre conseguir atingir o objetivo de cada iteração. A velocidade informa a equipe e o cliente da capacidade de entregar software funcionando num ritmo constante e sustentável.

- **Quando parar de utilizar?**: Quando o projeto acabar ou quando, após atingido o período de estabilização, a velocidade se tornar "conhecida" pela equipe e pelo cliente.

- **Formas de Manipulação**: Estimar mais pontos para o mesmo trabalho fará com que a velocidade aumente, por isso ela não pode ser usada para comparar diferentes equipes. Enquanto uma equipe estima 1000 pontos, uma outra pode estimar 600 pontos para fazer o mesmo trabalho, tornando a comparação inviável.

- **Cuidados e Observações**: Velocidade não representa valor agregado. Um time pode ter excelente velocidade, entregando software freqüentemente durante um certo tempo, porém sem trazer o retorno (financeiro ou não) esperado. Comparar a velocidade de equipes diferentes também pode ser um problema (conforme discutido acima). Da mesma forma, medir a velocidade por membro da equipe também pode gerar conflitos e atrapalhar. Ela deve sempre ser medida no nível da equipe.

Fator de Integração

- **Classificação**: Quantitativa e objetiva.
- **Pergunta**: Quanto tempo a equipe demora para integrar com o repositório? Quanto código é alterado antes de ser integrado no repositório?
- **Base de Medição**: O fator de integração IF_i para a iteração i é calculado da seguinte maneira:

$$IF_i = \frac{LA_i + LR_i + LU_i}{TC_i}$$

onde:

LA_i = número total de linhas adicionadas na iteração i

LR_i = número total de linhas removidas na iteração i

LU_i = número total de linhas atualizadas na iteração i

TC_i = número total de *commits* na iteração i

- **Suposições**: A equipe utiliza um sistema de controle de versão, onde o código é integrado freqüentemente pelos integrantes da equipe.
- **Tendência Esperada**: Se a equipe está praticando a Integração Contínua de forma correta, o fator de integração deve ser baixo, indicando que há poucas linhas alteradas a cada *commit*. Dessa forma, a tendência esperada para uma equipe que está aprendendo a Integração Contínua é de diminuição.
- **Quando utilizar?**: Quando os membros da equipe estiverem demorando muito tempo para sincronizar ou integrar código novo no repositório ou quando as alterações forem muito grandes.
- **Quando parar de utilizar?**: Assim que o fator de integração chegar em um nível aceitável e a equipe deixar de ter problemas de sincronização de trabalho.

- **Formas de Manipulação**: Essa métrica pode ser manipulada se os integrantes da equipe resolverem fazer um *commit* no repositório a cada pequena alteração, aumentando muito a quantidade de *commits*. Isso deve ser desencorajado pois toda alteração no repositório deve mantê-lo num estado consistente, ou seja, os testes automatizados devem continuar passando.

- **Cuidados e Observações**: Essa métrica não leva em conta a qualidade do código que está sendo integrado ao repositório. Conforme discutido acima, um grande aumento no número de *commits* pode piorar a Integração Contínua e deixar o repositório num estado inconsistente. Manter a qualidade do código através de uma bateria de testes automatizados deve ser uma preocupação constante da equipe.

Fator de Teste

- **Classificação**: Quantitativa e objetiva.

- **Pergunta**: Qual a relação entre código de teste e código de produção que está sendo produzido pela equipe?

- **Base de Medição**: O fator de teste T_i para a iteração i é calculado como a razão entre o número de linhas de código de teste e o número de linhas de código de produção:

$$T_i = \frac{TLOTC_i}{TLOC_i}$$

onde:

$TLOTC_i$ = número total de linhas de código de teste na iteração i
$TLOC_i$ = número total de linhas de código de produção na iteração i

- **Suposições**: A equipe está desenvolvendo código de produção e código de testes autom-

atizados para verificar a qualidade do software produzido.

- **Tendência Esperada**: Se uma equipe utiliza técnicas como TDD desde o início do projeto, o fator de testes será alto (em muitos casos inclusive com valor acima de 1). Para equipes onde os testes não são desenvolvidos junto com o código de produção, o fator de testes será provavelmente baixo e a tendência é de melhoria.

- **Quando utilizar?**: Quando a equipe estiver preocupada com a qualidade do software produzido, ou estiver monitorando a melhoria de práticas como o Desenvolvimento Dirigido por Testes, a Integração Contínua, o Design Incremental ou a Refatoração.

- **Quando parar de utilizar?**: Assim que a qualidade do software produzido atingir o nível de qualidade esperado.

- **Formas de Manipulação**: Essa métrica pode ser manipulada através da produção de muito código de teste que não verifica o comportamento esperado do sistema (testes sem *assert*, por exemplo). Da mesma forma, medir linhas de código sempre pode gerar um incentivo para tornar o código menos legível, evitando quebras de linha que afetariam essa métrica.

- **Cuidados e Observações**: Um fator de teste maior que 1 não representa necessariamente o cenário ideal, assim como não garante que o código está totalmente testado ou com boa qualidade. Apesar de influenciar a qualidade do software produzido, essa métrica não serve como medida objetiva de qualidade.

4. Estudo de Caso

Conforme discutido no Capítulo 2, a abordagem dos Métodos Ágeis é baseada num conjunto de práticas simples que geram *feedback* freqüente para que os membros da equipe entendam o andamento atual do projeto e para guiá-los em direção a um ambiente de melhoria contínua. Para auxiliar o *tracker* e a equipe nessa tarefa, o Capítulo 3 apresentou diversas medidas e métricas.

Este capítulo descreve um estudo de caso empírico baseado em sete projetos de desenvolvimento de software utilizando Métodos Ágeis, principalmente XP. O autor deste texto participou desses projetos durante o primeiro semestre de 2006, atuando como *coach* e monitor de cinco projetos na Universidade de São Paulo (USP) e como consultor em dois projetos governamentais na Assembléia Legislativa do Estado de São Paulo (ALESP).

O principal objetivo deste estudo de caso é avaliar o papel das métricas de acompanhamento nesses projetos. Além disso, este estudo classifica os sete projetos em relação aos termos e categorias propostos pelo *Extreme Programming Evaluation Framework* (XP-EF) [112, 113].

4.1. Método e Métricas

Esta seção apresenta os métodos de coleta e as métricas utilizadas para analisar os projetos descritos na Seção 4.2. As métricas usadas neste estudo de caso são métricas de diagnóstico e foram apresentadas na Seção 3.5. As métricas quantitativas e objetivas foram coletadas de forma automatizada das seguintes fontes:

- **Plug-in do Eclipse**: métricas como TLOC, $v(G)$, WMC, LCOM, DIT, NOC, AC e EC foram coletadas diretamente do código-fonte através do plug-in Eclipse Metrics[1]. Os arquivos foram obtidos do repositório de código, nas revisões correspondentes ao final de cada iteração analisada. O plug-in então exportava um arquivo XML com os dados brutos de cada métrica. Esse arquivo XML foi então pós-processado por um script em Ruby para filtrar dados de produção (excluindo código de teste quando pertinente) e para gerar as estatísticas finais de cada métrica.

- **Repositório de Código**: métricas como o número de *commits* e número de linhas alteradas por *commit* foram obtidas diretamente do repositório (CVS e Subversion), através de um script em Bash que filtrava os dados necessários, analisando o histórico, *logs* e *diffs* do repositório. Essas métricas foram coletadas em relação ao final de cada iteração.

- **XPlanner**: métricas como estimativas e número de histórias entregues foram coletadas em relação ao final de cada iteração diretamente do XPlanner [78], uma ferramenta Web leve para planejamento e controle de projetos de XP. Vale ressaltar que as estimativas têm caráter subjetivo no XPlanner, pois seu verdadeiro significado pode variar de equipe para equipe. Dessa forma, essas métricas não podem ser utilizadas para comparação entre diferentes projetos.

Além disso, outra métrica quantitativa e objetiva sugerida pelo XP-EF para avaliar a produtividade da equipe foi coletada. O **Parâmetro de Produtividade de Putnam** (PPP) é uma métrica baseada na análise histórica de diversos projetos, utilizando algumas medidas apresentadas na Seção 3.5.1 e calculada da seguinte maneira [87]:

$$PPP = \frac{TLOC}{(\text{Esforço}/\beta)^{\frac{1}{3}}\,(\text{Tempo})^{\frac{4}{3}}}$$

[1] http://metrics.sourceforge.net

$TLOC$ é o total de linhas de código, Esforço é o número de pessoas-mês de trabalho realizado no projeto, β é um fator escolhido a partir de uma tabela construída por Putnam com base em dados de produção obtidos de diversos projetos de software [87] e Tempo é o número de meses gastos no projeto. Outras métricas quantitativas e subjetivas foram coletadas a partir das seguintes fontes:

- **Questionários**: Foi utilizado um questionário desenvolvido por Krebs [70], que foi adaptado para coletar informações sobre o nível educacional e de experiência profissional da equipe, o grau de utilização de cada prática de XP (veja o Apêndice A) e uma nota geral (numa escala de 0 a 10) sobre a qualidade do *tracking* na equipe. Para cada questão, os membros deveriam dar uma nota para o nível "atual" e "desejado" da equipe em relação a cada prática de XP. O questionário foi aplicado ao final da última iteração do primeiro semestre de 2006.

- **Calendário *Niko-Niko***: Durante a última iteração, também foram coletadas informações sobre a moral de equipe, pedindo para que cada participante atualizasse um *calendário niko-niko* [81] semanalmente. Ao final de cada dia de trabalho, os membros da equipe deveriam colar um adesivo no calendário, com uma cor que indicasse sua motivação (feliz, indiferente ou infeliz). Esse calendário ficou conhecido como "Humorômetro" em algumas equipes, pois representa em cores o humor de cada membro da equipe durante os dias da semana. A natureza dessa métrica, originalmente qualitativa, foi codificada para um valor quantitativo entre 0% e 100%, considerando um adesivo verde (feliz) como 100%, um amarelo (indiferente) como 50% e um vermelho (infeliz) como 0%. Essa métrica corresponde a uma das Medidas de Resultado (XP-om) sugeridas pelo XP-EF.

Por fim, métricas qualitativas e subjetivas foram obtidas através de **Entrevistas** realizadas ao final da última iteração do primeiro semestre de 2006. Foram realizadas entrevistas semi-estruturadas [99], com uma mistura de perguntas abertas e específicas, para auxiliar no entendimento e dar uma visão geral do que acontecia em cada projeto.

4.2. Projetos

Esta seção apresenta os sete projetos de desenvolvimento de software que foram analisados. Cinco deles foram conduzidos em ambiente acadêmico durante um semestre inteiro, no curso de Programação Extrema da Universidade de São Paulo [49]. Os outros dois foram conduzidos numa instituição governamental, a Assembléia Legislativa do Estado de São Paulo (ALESP).

4.2.1. Formato de Apresentação

A maioria dos projetos, exceto o último, seguiram a maioria das práticas de XP, então eles serão descritos nos termos e categorias do *Extreme Programming Evaluation Framework* (XP-EF) [112, 113]. Esse arcabouço descreve o contexto do estudo de caso, a extensão da adoção das práticas de XP e os resultados dessa adoção. O XP-EF é composto de três partes: Fatores de Contexto (*Context Factors* ou XP-cf), Métricas de Aderência a XP (*XP Adherence Metrics* ou XP-am) e Medidas do Resultado de XP (*XP Outcome Measures* ou XP-om).

Os Fatores de Contexto (XP-cf) e as Medidas de Resultado (XP-om) serão descritas nessa seção. As Métricas de Aderência (XP-am) foram coletadas através de um questionário subjetivo, construído com base em [70] e apresentado no Apêndice A.

Alguns dos Fatores de Contexto (XP-cf) são comuns entre todos os projetos e serão descritos na Seção 4.2.3. Fatores que diferem serão apresentados para cada projeto no seguinte formato:

- **Nome do Projeto**
- **Descrição**: uma breve descrição dos objetivos do projeto.
- **Classificação do Software**: conforme sugerido por Jones [66].
- **Tabela de Informações**: uma tabela descrevendo os fatores sociológicos e específicos do projeto. Informações como tamanho da equipe, níveis educacional e experiência profissional foram coletadas a partir dos resultados do questionário (Apêndice A). Conhecimento de domínio e da linguagem foram analisados subjetivamente pelo autor. Dados

quantitativos como o número de histórias entregues e KLOEC foram coletados conforme o método apresentado na Seção 4.1.

- **Gráfico Polar**: um gráfico polar de 5 eixos descrevendo os fatores de desenvolvimento propostos por Boehm e Turner [24]. Os eixos representam fatores de risco para avaliar o quão apropriado seria a adoção das práticas ágeis. Os fatores de risco, conforme apresentados na Seção 3.5.1, são: tamanho da equipe, criticalidade, dinamismo de requisitos, pessoal (experiência da equipe) e cultural (aceitação das mudanças). Segundo Boehm e Turner, quando os pontos de um projeto são ligados, o formato da figura provê informação visual. Formatos em direção ao centro do gráfico sugerem o uso de um Método Ágil. Formatos em direção à periferia do gráfico sugerem o uso de um método com mais ênfase no planejamento (*plan-driven*). Formatos mais variados sugerem o uso de um método híbrido [24].

- **Medidas de Resultado (XP-om)**: uma tabela descrevendo as Medidas de Resultado disponíveis (XP-om). Métricas de produtividade foram diretamente calculadas a partir das medidas disponíveis, como KLOEC, pessoas-mês (*person-month* ou PM), número de histórias entregues e Parâmetro de Produtividade de Putnam (PPP). Além das métricas de produtividade, foi coletada uma métrica subjetiva da moral da equipe, através de um *calendário niko-niko* [81], descrito na Seção 4.1.

4.2.2. Considerações

Os projetos acadêmicos tinham um horário de trabalho diferente dos projetos governamentais. A cada semana, os estudantes deveriam comparecer ao laboratório para duas sessões, uma de duas e outra de três horas de duração. Além dessas duas sessões obrigatórias, era sugerido que os estudantes viessem ao laboratório para mais quatro horas extras de trabalho por semana. Apesar dessas horas não serem controladas pelos professores, muitos estudantes compareciam ao laboratório para cumprir as horas extras. Cada estudante trabalhava, em

média, de 6 a 8 horas por semana. O horário de trabalho para os projetos governamentais era diferente: cada membro da equipe devia trabalhar no projeto um total de 30 horas por semana.

Na academia, o semestre inteiro é considerado como uma *release*, que é desenvolvida em duas a quatro iterações. A recomendação era que os times trabalhassem em iterações com duração máxima de 1 mês, mas de acordo com a experiência do time nas tecnologias utilizadas e com o calendário escolar do semestre, a duração exata de cada iteração variava de time para time.

Além das práticas adaptadas apresentadas na Seção 2.9.7, uma outra prática foi utilizada. Os projetos descritos nesse capítulo eram geralmente compostos por times sem experiência prévia com Métodos Ágeis e, muitas vezes, com as tecnologias, arcabouços e ferramentas utilizadas. Uma diferença muito acentuada de experiência entre os membros da equipe pode prejudicar sua produtividade [111]. Na tentativa de diminuir essa diferença e equalizar o nível técnico da equipe, uma fase de **Treinamento** foi conduzida no início dos projetos. Durante essa fase, os times tinham aulas sobre as práticas de XP, tecnologias, ferramentas e arcabouços que seriam utilizados durante o projeto, como: programação orientada a objetos, coleções Java, Struts, testes de unidade, testes de aceitação, refatoração, integração contínua e controle de versões com CVS.

Por fim, não foi possível coletar dados históricos relacionados a defeitos e *bugs*. No momento em que os dados foram coletados, nenhum dos projetos estava implantado em produção ainda, exceto pelo Projeto 7 (descrito na Seção 4.2.5). O Projeto 7 não possui um sistema para gerenciamento de *bugs* e já estava implantado quando as primeiras práticas de XP foram introduzidas à equipe.

4.2.3. Fatores de Contexto em Comum (XP-cf)

Alguns dos Fatores de Contexto (XP-cf) como: ergonômico, tecnológico e geográfico, são semelhantes para todos os projetos. Eles são detalhados na Tabela 4.1 pare evitar duplicação de informação.

Fatores Ergonômicos	
Disposição Física	Espaço de trabalho arranjado num espaço amplo e aberto.
Nível de Distração no Espaço de Trabalho	Moderado, com outros times trabalhando perto no mesmo ambiente.
Comunicação com o Cliente	Geralmente o cliente está presente ou trabalha no mesmo prédio. Ocasionalmente por e-mail.
Fatores Tecnológicos	
Método de Desenvolvimento	Principalmente XP (exceto no Projeto 7, discutido na Seção 4.2.5).
Linguagem	Java, com tecnologias e arcabouços relacionados (Struts, JDBC, JSF, Swing, etc.).
Ferramentas	Eclipse, CheckStyle, CVS/Subversion, XPlanner, xUnit, Wiki.
Fatores Geográficos	
Localização do Time	Todos trabalham juntos no mesmo ambiente.
Quantidade de Clientes e Localização	Academia: 1 cliente presente. Governo: 1 representante do cliente que trabalha no mesmo prédio.

Tabela 4.1.: Fatores Ergonômicos, Tecnológicos e Geográficos (XP-cf)

4.2.4. Projetos Acadêmicos

Projeto 1 – "Archimedes" – http://archimedes.incubadora.fapesp.br

- **Descrição**: Um software de código aberto para CAD (*computer-aided design*) com ênfase nas necessidades de um arquiteto profissional. Esse projeto começou alguns meses antes da aula de XP, quando os estudantes envolvidos começaram a recrutar o time e a fazer um levantamento inicial de requisitos com arquitetos profissionais. Foram analisadas as

primeiras 4 iterações do projeto, durante o primeiro semestre de 2006.

- **Classificação do Software**: Software comercial, desenvolvido como um projeto de código aberto.

- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.2(a), 4.2(c) e na Figura 4.2(b).

Projeto 2 – "GVC" – Grid Video Converter

- **Descrição**: Uma aplicação Web que oferece aos usuários uma grade computacional para conversão de arquivos de vídeos entre diversos formatos, codificações e diferentes qualidades. O arquivo é enviado por um usuário registrado, convertido no servidor e, após convertido, fica disponível para *download*. Foram analisadas as primeiras 3 iterações do projeto, durante o primeiro semestre de 2006.

- **Classificação do Software**: Software para usuário final.

- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.3(a), 4.3(c) e na Figura 4.3(b).

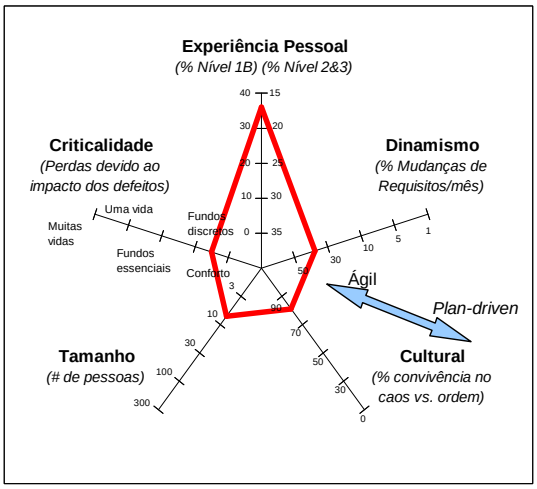
Projeto 3 – "Colméia" – http://colmeia.incubadora.fapesp.br

- **Descrição**: Um sistema completo para gerenciamento de bibliotecas desenvolvido na Universidade de São Paulo por estudantes de graduação e pós-graduação durante as quatro últimas turmas da disciplina de XP. Foi analisada a implementação de um novo módulo que permite ao usuário a busca de qualquer item no acervo da biblioteca. Já havia sido desenvolvido grande parte do modelo do banco de dados, assim como outros

Fatores Sociológicos	
Tamanho da Equipe	8
Nível de Educação da Equipe	Estudantes de Graduação: 8
Nível de Experiência da Equipe	< 5 anos: 8
Conhecimento do Domínio	Baixo
Conhecimento da Linguagem	Alto

Fatores Específicos do Projeto	
Histórias Entregues	64
Domínio	Aplicativo *Stand-alone*, CAD
Pessoas-Mês	6,4
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	18,832
Milhares de Linhas de Código (KLOEC) do Sistema	18,832

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	2,943
Histórias/PM	10
Parâmetro de Produtividade de Putnam	0,902
Moral da Equipe	97,63 %

(c) Medidas de Resultado Disponíveis (XP-om)

Tabela 4.2.: XP-cf e XP-om do Projeto 1 – Archimedes

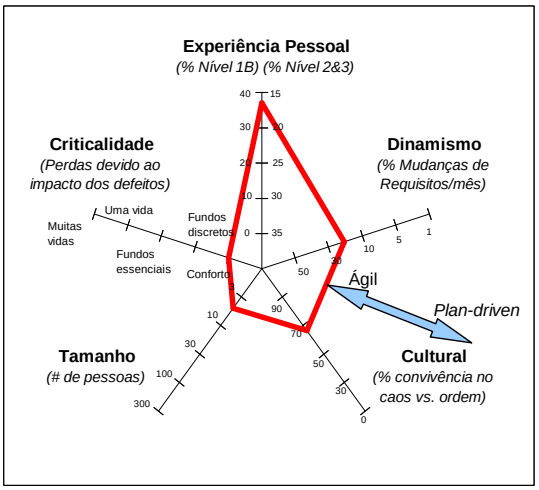
módulos do sistema. Por essa razão, antes de começar a codificação do novo módulo, a equipe passou um certo tempo estudando e entendendo o código e o modelo de banco de dados existente.

- **Classificação do Software**: Sistema de informação.
- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.4(a), 4.4(c)

Fatores Sociológicos	
Tamanho da Equipe	6 (1 substituído durante o projeto)
Nível de Educação da Equipe	Estudantes de Graduação: 2 Estudantes de Pós-Graduação: 4
Nível de Experiência da Equipe	< 5 anos: 4 5–10 anos: 2
Conhecimento do Domínio	Baixo
Conhecimento da Linguagem	Alto

Fatores Específicos do Projeto	
Histórias Entregues	16
Domínio	Web, Computação em Grade, Conversão de Vídeo
Pessoas-Mês	4,2
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	2,535
Milhares de Linhas de Código (KLOEC) do Sistema	2,535

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	0,604
Histórias/PM	3,810
Parâmetro de Produtividade de Putnam	0,134
Moral da Equipe	64 %

(c) Medidas de Resultado Disponíveis (XP-om)

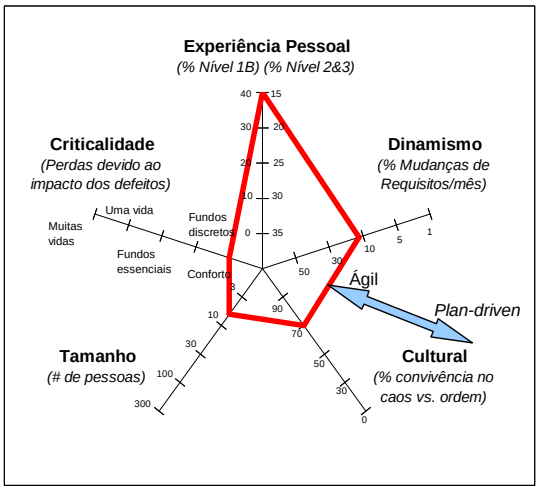
Tabela 4.3.: XP-cf e XP-om do Projeto 2 – GVC

e na Figura 4.4(b).

Fatores Sociológicos	
Tamanho da Equipe	7 − 1 (deixou a equipe)
Nível de Educação da Equipe	Estudantes de Graduação: 7
Nível de Experiência da Equipe	< 5 anos: 6 5–10 anos: 1
Conhecimento do Domínio	Baixo
Conhecimento da Linguagem	Moderado

Fatores Específicos do Projeto	
Histórias Entregues	12
Domínio	Web, Sistema para Gerenciamento da Biblioteca
Pessoas-Mês	4,2
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	8,067
Milhares de Linhas de Código (KLOEC) do Sistema	31,252

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	1,921
Histórias/PM	2,857
Parâmetro de Produtividade de Putnam	0,427
Moral da Equipe	73 %

(c) Medidas de Resultado Disponíveis (XP-om)

Tabela 4.4.: XP-cf e XP-om do Projeto 3 – Colméia

Projeto 4 – "GinLab" – Ginástica Laboral

- **Descrição**: Um aplicativo *stand-alone* para auxiliar na recuperação e prevenção das Lesões por Esforço Repetitivo (LER). O programa alerta o usuário freqüentemente para lembrá-lo de fazer pausas e realizar algumas rotinas pré-configuradas de exercícios. Foram analisadas as primeiras 3 iterações do projeto, durante o primeiro semestre de

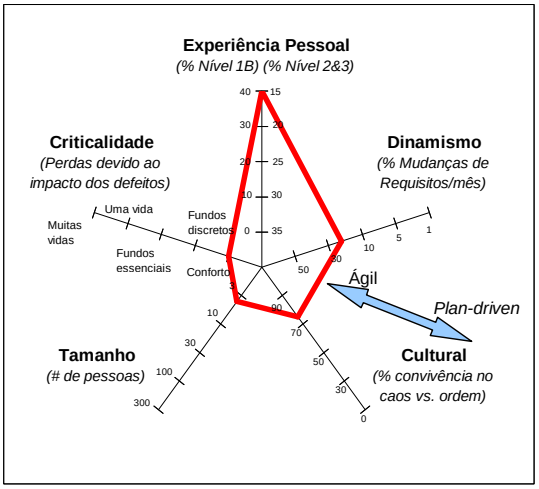
2006.

- **Classificação do Software**: Software para usuário final.
- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.5(a), 4.5(c) e na Figura 4.5(b).

Fatores Sociológicos	
Tamanho da Equipe	4
Nível de Educação da Equipe	Estudantes de Graduação: 4
Nível de Experiência da Equipe	< 5 anos: 4
Conhecimento do Domínio	Baixo
Conhecimento da Linguagem	Moderado

Fatores Específicos do Projeto	
Histórias Entregues	18
Domínio	Aplicativo *Stand-alone*, Web
Pessoas-Mês	2,6
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	4,721
Milhares de Linhas de Código (KLOEC) do Sistema	4,721

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	1,816
Histórias/PM	6,923
Parâmetro de Produtividade de Putnam	0,294
Moral da Equipe	72,33 %

(c) Medidas de Resultado Disponíveis (XP-om)

Tabela 4.5.: XP-cf e XP-om do Projeto 4 – Ginástica Laboral

Projeto 5 – "Borboleta" – http://borboleta.incubadora.fapesp.br

- **Descrição**: Um sistema de computação móvel para apoio a projetos de telemedicina projetado para auxiliar os profissionais da saúde do programa Atenção Primária Domiciliar (APD) do Centro de Saúde-Escola na Faculdade de Medicina da USP [93]. O sistema implementa funcionalidade para: marcação de consultas domiciliares, prontuário digital, visualização e consulta de medicamentos e é composto por três partes: um aplicativo J2ME executado em dispositivos móveis utilizados pelos médicos e enfermeiras nas visitas, um aplicativo *desktop* que sincroniza as informações entre a base de dados do centro de saúde e as aplicações nos dispositivos móveis e um aplicativo Web que serve de meio de comunicação entre o aplicativo móvel e o aplicativo *desktop*. O projeto se iniciou em 2005 como projeto de conclusão de curso de três alunos de graduação e, durante o primeiro semestre de 2006, novas funcionalidades foram implementadas no laboratório de XP. Foram analisadas as primeiras 3 iterações do projeto, durante a segunda fase do projeto na disciplina de XP.

- **Classificação do Software**: Sistema de informação.

- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.6(a), 4.6(c) e na Figura 4.6(b).

4.2.5. Projetos Governamentais

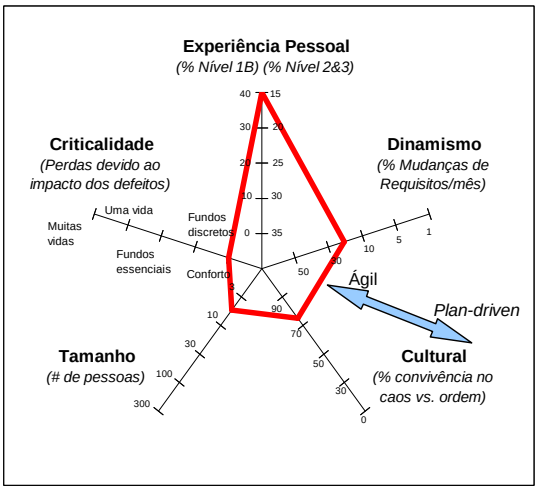
Projeto 6 – "Chinchilla"

- **Descrição**: Um sistema de recursos humanos para gerenciar as informações de todos os empregados de uma instituição do governo. O sistema está sendo desenvolvido na Assembléia Legislativa do Estado de São Paulo (ALESP) por um grupo de empregados da assembléia e guiados por consultores da Universidade de São Paulo. Conforme discutido

Fatores Sociológicos	
Tamanho da Equipe	6
Nível de Educação da Equipe	Estudantes de Graduação: 1 Estudantes de Pós-Graduação: 5
Nível de Experiência da Equipe	< 5 anos: 2 5–10 anos: 4
Conhecimento do Domínio	Baixo
Conhecimento da Linguagem	Alto

Fatores Específicos do Projeto	
Histórias Entregues	24
Domínio	Computação Móvel, Web, Sistema de Telemedicina
Pessoas-Mês	4,2
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	7,753
Milhares de Linhas de Código (KLOEC) do Sistema	15,444

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	1,846
Histórias/PM	5,714
Parâmetro de Produtividade de Putnam	0,411
Moral da Equipe	79,75 %

(c) Medidas de Resultado Disponíveis (XP-om)

Tabela 4.6.: XP-cf e XP-om do Projeto 5 – Borboleta

na Seção 4.2.2, algum tempo foi gasto durante um treinamento para a equipe nas tecnologias e no Método Ágil adotado para o projeto (XP). Após algumas iterações, o nível de participação dos consultores foi diminuindo, permitindo que o time de funcionários gradualmente começasse a guiar o projeto. Foram analisadas 8 iterações da primeira *release* do sistema.

- **Classificação do Software**: Sistema de informação.

- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.7(a), 4.7(c) e na Figura 4.7(b).

Projeto 7 – "SPL" – Sistema do Processo Legislativo

O desenvolvimento inicial desse sistema foi terceirizado para uma empresa privada contratada. Após 2 anos de desenvolvimento, o sistema foi implantado e o time de funcionários da ALESP recebeu treinamento para dar suporte e manutenção no sistema. Devido à falta de experiência nas tecnologias utilizadas para construir o sistema e ao grande número de defeitos que surgiram quando o sistema começou a ser utilizado em produção, o time passava por sérias dificuldades para dar suporte aos usuários finais, consertar defeitos e implementar novas funcionalidades. Quando os consultores da Universidade de São Paulo foram chamados para auxiliar no projeto, algumas práticas ágeis foram introduzidas para ajudá-los a lidar com o sistema, como: Integração Contínua [44], Testes (testes de unidade e de aceitação automatizados), Refatoração [45] e Área de Trabalho Informativa.

Além disso, conforme descrito na Seção 4.2.2, treinamentos esporádicos foram ministrados durante alguns meses sobre as tecnologias e tópicos de interesse que ajudariam a equipe no trabalho do dia-a-dia, como: programação orientada a objetos, coleções Java, Struts, testes de unidade, testes de aceitação, refatoração, integração contínua e controle de versões com CVS.

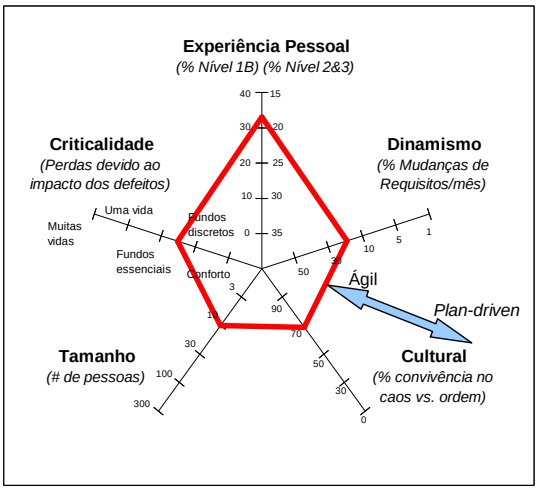
- **Descrição**: Um sistema de *workflow* para auxiliar os deputados e seus assessores no gerenciamento dos documentos legais (proposituras, leis, atas, etc.) durante o processo legislativo. Foram analisados os primeiros 3 meses de projeto após a introdução das práticas ágeis durante o primeiro semestre de 2006.

- **Classificação do Software**: Sistema de informação.

Fatores Sociológicos	
Tamanho da Equipe	9 ± 1
Nível de Educação da Equipe	Graduados: 8 Estudantes de Graduação (estagiários): 1
Nível de Experiência da Equipe	< 5 anos: 2 5–10 anos: 8
Conhecimento do Domínio	Alto
Conhecimento da Linguagem	Baixo

Fatores Específicos do Projeto	
Histórias Entregues	106
Domínio	Web, Sistema de Recursos Humanos, Sistema Governamental
Pessoas-Mês	58,5
Meses de Projeto Analisados	11
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	48,517
Milhares de Linhas de Código (KLOEC) do Sistema	48,517

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	0,829
Histórias/PM	1,812
Parâmetro de Produtividade de Putnam	0,367
Moral da Equipe	67,48 %

(c) Medidas de Resultado Disponíveis (XP-om)

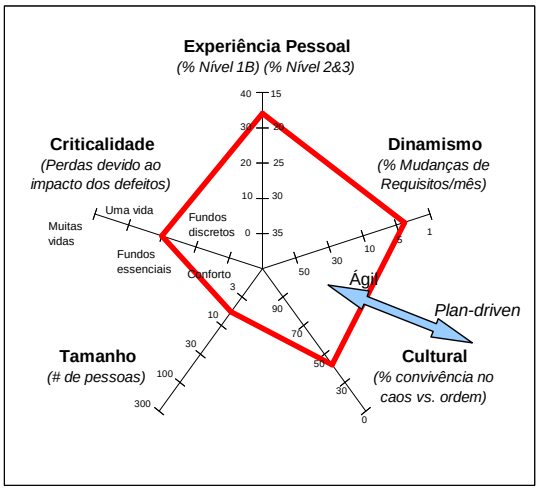
Tabela 4.7.: XP-cf e XP-om do Projeto 6 – Chinchilla

- **Tabela de Informações**: Os Fatores de Contexto (XP-cf), as Medidas de Resultado (XP-om) e o Gráfico Polar são apresentados, respectivamente, nas Tabelas 4.8(a), 4.8(c) e na Figura 4.8(b).

Fatores Sociológicos	
Tamanho da Equipe	5 empregados em tempo integral + 1 consultor em tempo parcial + 2 estagiários em tempo parcial
Nível de Educação da Equipe	Estudantes de Graduação: 1 Graduados: 7
Nível de Experiência da Equipe	< 5 anos: 1 5–10 anos: 7
Conhecimento do Domínio	Alto
Conhecimento da Linguagem	Baixo

Fatores Específicos do Projeto	
Histórias Entregues	N/A
Domínio	Web, *Workflow*
Pessoas-Mês	15
Meses de Projeto Analisados	4
Milhares de Linhas de Código (KLOEC) Novas & Alteradas	6,819
Milhares de Linhas de Código (KLOEC) do Sistema	177,016

(a) Fatores Sociológicos e Específicos de Projeto (XP-cf)



(b) Fatores de Desenvolvimento (XP-cf)

Medidas de Resultado	
Produtividade	
KLOEC/PM	0,455
Histórias/PM	N/D
Parâmetro de Produtividade de Putnam	0,236
Moral da Equipe	N/D

(c) Medidas de Resultado Disponíveis (XP-om)

Tabela 4.8.: XP-cf e XP-om do Projeto 7 – SPL

4.2.6. Gráfico em Radar de XP (XP Radar Chart)

A partir do questionário adaptado de Krebs [70] (Apêndice A) é possível avaliar os diferentes níveis de adoção das práticas de XP em cada projeto. Para isso, o Gráfico em Radar de XP

proposto por Wake [108] é um bom indicador visual. Wake propõe um gráfico com cinco eixos, representando as dimensões de uma implementação de XP: programação, planejamento, cliente, pareamento e equipe. O valor de cada eixo do gráfico representa a média das práticas correspondentes, obtidas a partir das respostas do questionário e arredondadas para o inteiro mais próximo para melhorar a legibilidade da figura. Algumas das práticas são consideradas em mais de um eixo do gráfico, conforme descrito na Tabela 4.9. Eixos que estão com valores mais próximos ao centro do gráfico representam uma deficiência na adoção de XP. A Figura 4.1 apresenta o gráfico para os sete projetos apresentados neste capítulo.

Eixo do Radar	**Práticas de XP**
Programação	Desenvolvimento Dirigido por Testes, Refatoração e Design Simples
Planejamento	Versões Pequenas, Jogo do Planejamento, Ritmo Sustentável, Lições Aprendidas e *Tracking*
Cliente	Desenvolvimento Dirigido por Testes, Jogo do Planejamento e Cliente com os Desenvolvedores
Pareamento	Programação Pareada, Integração Contínua e Propriedade Coletiva do Código
Equipe	Integração Contínua, Desenvolvimento Dirigido por Testes, Padronização de Código, Metáfora e Lições Aprendidas

Tabela 4.9.: Mapeamento entre práticas e eixos (algumas aparecem em mais de um eixo)

4.3. Análise dos Resultados

Esta seção analisa o papel das métricas de acompanhamento utilizadas nos projetos descritos na Seção 4.2. Serão discutidos também o resultado das entrevistas, do questionário apresentado no Apêndice A assim como outros assuntos relacionados à classificação dos projetos no XP-EF.

4.3.1. Análise das Respostas do Questionário de Aderência

A Figura 4.2 apresenta os resultados do questionário adaptado de Krebs [70], agregados por prática e por projeto.

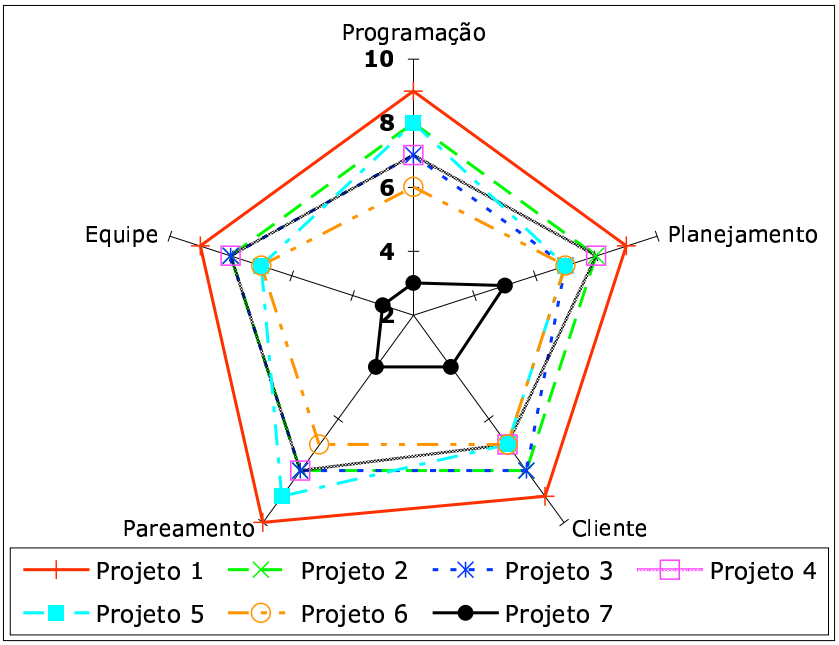


Figura 4.1.: Gráfico em Radar de XP

Ao analisar as respostas, percebe-se que a média das notas "desejadas" supera as notas "atuais" em todas as práticas para todos os projetos. Isso significa que todos as equipes achavam que podiam melhorar alguma coisa em todas as práticas de XP. O questionário foi respondido por 48 indivíduos e houve apenas uma resposta cujo valor desejado era inferior ao valor atual (na prática Design Simples).

Foi constatado também um resultado semelhante ao reportado por Krebs, autor do questionário original [70]: a média das notas desejadas para cada prática de XP era maior que a média desejada para a nota geral de XP. Isso, de acordo com Krebs, indica um maior entusiasmo da equipe com as técnicas mais práticas e menos com os valores e princípios de XP. A Programação Pareada no Projeto 7 foi uma exceção a esse comportamento, demonstrando sua resistência contra essa prática. O Projeto 5 também teve algumas notas desejadas menores que a nota geral de XP. Nesse caso eles tinham grande expectativa em relação ao processo como um todo ao invés das práticas individuais, tendo, inclusive, a maior das notas gerais de XP entre todos os projetos (9.7). O Projeto 5, inclusive, foi o único a apresentar uma

		Projeto 1		Projeto 2		Projeto 3		Projeto 4		Projeto 5		Projeto 6		Projeto 7	
		Média	D.P.	Média	D.P	Média	D.P	Média	D.P.	Média	D.P.	Média	D.P.	Média	D.P.
Nível Educacional		3.9	0.3	4.5	0.8	3.5	0.5	3.3	0.4	5.2	0.9	3.6	2.0	5.0	1.0
Nível de Experiência		1.0	1.6	4.0	3.7	2.2	2.2	0.8	1.3	4.5	3.3	7.4	3.3	7.6	3.3
Programação Pareada	Atual	9.9	0.2	7.2	1.5	8.7	0.9	9.5	0.9	8.8	0.7	6.3	1.6	3.1	1.2
	Desejado	10.0	0.0	9.3	0.7	10.0	0.0	10.0	0.0	9.8	0.4	8.6	0.9	7.6	0.7
Versões Pequenas	Atual	8.8	0.6	8.0	1.0	6.2	0.4	8.5	0.9	7.5	0.8	6.2	1.4	6.5	1.2
	Desejado	9.3	0.7	9.7	0.7	10.0	0.0	9.3	0.8	9.0	1.0	8.5	0.8	8.3	1.2
Integração Contínua	Atual	10.0	0.0	8.8	0.9	8.8	0.9	7.8	0.4	9.8	0.4	7.4	1.6	2.9	2.1
	Desejado	10.0	0.0	10.0	0.0	9.0	1.0	9.8	0.4	10.0	0.0	9.0	1.3	7.8	2.9
Desenvolvimento Dirigido por Testes	Atual	8.5	0.5	6.8	0.9	8.5	1.8	7.0	1.4	6.5	1.0	7.2	1.3	3.3	1.6
	Desejado	9.3	0.7	9.3	0.7	9.0	1.5	9.3	0.9	9.3	0.7	8.9	1.3	8.6	1.1
Jogo do Planejamento	Atual	9.1	0.8	8.8	1.0	7.6	1.2	8.3	0.4	7.8	0.7	7.4	2.7	4.6	3.1
	Desejado	10.0	0.0	9.5	0.8	10.0	0.0	9.0	1.0	9.5	0.8	9.3	0.9	9.1	1.0
Cliente Presente	Atual	8.0	0.5	8.3	1.1	8.3	0.7	6.5	0.9	5.5	0.8	6.4	1.7	5.6	2.3
	Desejado	9.6	0.7	10.0	0.0	9.0	1.0	9.5	0.9	9.3	0.9	9.1	0.9	8.6	1.1
Refatoração	Atual	9.8	0.4	8.8	1.8	5.7	0.9	5.8	1.3	8.3	1.7	5.6	1.5	3.3	3.4
	Desejado	10.0	0.0	10.0	0.0	10.0	0.0	9.3	0.8	10.0	0.0	9.9	0.3	9.0	1.8
Design Simples	Atual	9.3	0.8	8.4	1.0	7.3	0.9	7.0	1.2	7.8	1.2	5.7	1.8	2.6	1.3
	Desejado	9.8	0.7	9.5	0.8	8.7	2.2	9.0	0.7	10.0	0.0	9.0	1.1	8.6	1.1
Padronização de Código	Atual	9.1	0.6	8.2	0.9	6.8	0.9	8.8	1.6	6.5	1.6	6.0	1.8	1.6	0.7
	Desejado	9.9	0.3	10.0	0.0	10.0	0.0	9.8	0.4	10.0	0.0	9.6	0.8	9.6	1.0
Propriedade Coletiva do Código	Atual	8.8	1.3	6.5	1.1	6.8	0.9	6.8	1.9	7.5	1.1	6.0	1.6	4.6	1.6
	Desejado	10.0	0.0	9.3	0.9	9.7	0.7	9.0	1.0	9.5	0.8	9.1	1.0	9.1	1.4
Metáfora	Atual	8.6	1.3	8.4	1.0	5.7	1.4	8.0	0.0	7.0	1.6	6.6	1.5	3.6	1.6
	Desejado	10.0	0.0	10.0	0.0	9.3	0.9	8.8	0.8	10.0	0.0	9.9	0.3	9.4	0.9
Ritmo Sustentável	Atual	9.6	0.7	6.2	2.0	7.4	1.4	8.5	1.7	9.0	1.0	8.7	1.9	6.3	2.3
	Desejado	10.0	0.0	10.0	0.0	10.0	0.0	10.0	0.0	9.7	0.7	10.0	0.0	9.9	0.3
Lições Aprendidas	Atual	8.6	0.9	8.6	0.8	8.1	0.2	7.5	0.9	7.2	0.9	7.2	1.2	5.5	2.0
	Desejado	9.8	0.7	10.0	0.0	10.0	0.0	9.5	0.9	9.7	0.7	9.0	0.9	9.4	0.7
Tracking	Atual	9.1	0.7	6.0	1.3	4.7	1.4	8.5	0.9	5.3	0.9	7.6	1.0	1.8	2.5
	Desejado	10.0	0.0	10.0	0.0	9.7	0.7	9.8	0.4	9.3	1.5	9.9	0.3	9.3	0.8
Nota Geral de XP	Atual	8.1	0.3	6.7	0.7	7.0	0.9	7.5	0.5	7.5	1.0	6.9	1.3	4.3	1.8
	Desejado	9.0	0.9	9.0	1.0	8.7	0.9	8.8	0.4	9.7	0.7	8.4	0.8	8.1	1.8
Média Geral (exceto nota geral)	Atual	9.1		7.8		7.2		7.7		7.5		6.7		3.9	
	Desejado	9.8		9.8		9.6		9.4		9.7		9.3		8.9	

Figura 4.2.: Resultados do questionário adaptado de Krebs [70]

nota desejada geral de XP maior que a média geral das notas desejadas considerando todas as práticas (9.67 contra 9.65).

Por fim, outra informação que pode ser obtida dos resultados do questionário é a diferença entre as notas desejadas e atuais para cada prática. É possível perceber que as práticas no Projeto 1 estavam indo bem, com uma diferença máxima de 1.6 na prática Cliente Presente. O Projeto 7, por outro lado, estava começando a adotar algumas das práticas ágeis, demonstrando uma diferença muito maior na maioria das práticas. É possível notar diferenças de 8.0 e 7.5 para as práticas Padronização de Código e *Tracking*, respectivamente. Essas diferenças podem mostrar à equipe os principais pontos de melhoria, indicando quais as práticas que precisam

de mais atenção na próxima iteração.

4.3.2. Retrospectivas como Ferramenta de Acompanhamento

No questionário utilizado para coletar as métricas de aderência (XP-am), além da questão já existente sobre "Lições Aprendidas" (Retrospectivas), foi incluída uma questão sobre *tracking* (Vide Apêndice A). Conforme mostrado na Tabela 4.10, o desenvolvedor deve dar uma nota numa escala de 0 a 10 para o que considera o nível atual e o nível desejado do *tracking* na equipe, utilizando seis frases como exemplos da forma em que a prática foi implementada e como se manifesta na equipe. O desenvolvedor deve escolher a frase que mais se assemelha ao que o time faz (e deveria fazer) e dar a sua nota.

Tracking Atual:_____ Desejado:_____	**Existem diversos gráficos e informações espalhadas pelas paredes que nos ajudam a entender o andamento do projeto.** 10 Atualizamos as informações diariamente e descartamos os gráficos que não trazem informações relevantes. Eles nos ajudam a entender o andamento do projeto e identificar pontos de melhoria. 8 Temos alguns gráficos interessantes que são atualizados semanalmente. 6 As informações na parede são atualizadas ao final de cada *release*. 4 Os gráficos estão desatualizados e ninguém mais se importa com o que está na parede. Precisamos terminar o trabalho para entregar o sistema no prazo. 2 Não sei qual a utilidade dos gráficos na parede. Eles parecem não estar relacionados ao meu trabalho diário. Eu acho que se fossem retirados ninguém iria perceber. 0 Não temos nenhum gráfico na parede. Preferimos guardar informações importantes em arquivos e documentos que ficam armazenados em nosso repositório central. Quem tiver interesse pode atualizá-los ou consultá-los.

Tabela 4.10.: Questão adicionada ao questionário adaptado de [70]

A partir do resultado do questionário (apresentado na Seção 4.3.1), percebe-se que o *tracking* não estava funcionando bem em todas as equipes: enquanto os projetos 1, 4 e 6 tinham uma média de 9.1, 8.5 e 7.6, os projetos 2, 3, 5 e 7 tinham média de 6.0, 4.7, 5.3 e 1.8

respectivamente. Esse comportamento foi confirmado durante as entrevistas com os membros de cada projeto. Apesar disso, todos concordaram que a Retrospectiva foi uma prática muito valiosa para auxiliar a equipe no acompanhamento do projeto e na melhoria do processo. Os resultados das Retrospectivas eram afixados na Área de Trabalho Informativa e serviam como guias para a melhoria da equipe. Mesmo as equipes que não estavam indo tão bem no *tracking* se mostravam dispostas a seguir as melhorias propostas no cartaz da Retrospectiva. Pode-se concluir que a Retrospectiva é uma ferramenta muito valiosa para manter o time no caminho certo e para melhorar seu desempenho.

4.3.3. Nível Pessoal e Agilidade

Os fatores de desenvolvimento descritos na Seção 3.5.1 e apresentados na Seção 4.2 foram propostos por Boehm e Turner como um método baseado em riscos para classificar diferentes projetos entre a utilização de um Método Ágil ou um método com mais ênfase no planejamento (*plan-driven*) [24, 25]. Quando o polígono formado pelos pontos que representam os dados do projeto está próximo do centro do gráfico, sugere-se a adoção de um Método Ágil. Formatos mais voltados para a periferia do gráfico sugerem a adoção de um método com mais ênfase no planejamento (*plan-driven*).

Um dos fatores de risco propostos por Boehm e Turner é o nível pessoal de experiência com processos adaptativos. Este nível é medido de acordo com a escala proposta por Cockburn [34], e apresentada na Tabela 4.11. Esses níveis consideram a experiência que um integrante possui em adaptar o processo para situações particulares. Ao analisar os gráficos de cada projeto (Figuras 4.2(b), 4.3(b), 4.4(b), 4.5(b), 4.6(b), 4.7(b) e 4.8(b)), é possível observar que a experiência em adaptar o processo era similarmente baixa em todos os projetos. O vértice no topo de cada polígono está em direção à periferia em todos os gráficos. Isso, de acordo com o método de Boehm e Turner, sugere que um Método Ágil não seria apropriado para aqueles projetos. No entanto, uma avaliação subjetiva do autor revela que esse eixo não revela

adequadamente as diferenças entre os ambientes acadêmicos e governamentais.

Nível	**Características dos Membros da Equipe**
3	Capaz de reavaliar o método, quebrando suas regras para acomodar uma situação inesperada.
2	Capaz de adaptar o método para acomodar uma situação nova, porém previamente conhecida.
1A	Com treinamento, é capaz de realizar passos do método que exigem decisão como: re-estimar histórias, compor padrões, fazer refatorações compostas ou integração complexa com produtos COTS (*Commercial Off-the-Shelf* ou produtos comerciais de prateleira). Com experiência, pode se tornar um nível 2.
1B	Com treinamento, é capaz de realizar procedimentos simples do método como: codificar um método simples, fazer refatorações simples, seguir padrões de código e procedimentos de gerenciamento de configuração ou executar testes. Com experiência, pode aprender algumas habilidades do nível 1A.
-1	Pode ter habilidade técnica porém não é capaz ou não quer colaborar ou seguir o método proposto.

Tabela 4.11.: Nível pessoal de experiência proposto por Cockburn [34]

Ter pessoas com experiência em Métodos Ágeis e processos adaptativos é um importante fator de risco, porém elas geralmente não estarão disponíveis em grande quantidade. Conforme descrito na Seção 4.2.2, a abordagem diferenciada utilizada na adoção da Programação Extrema começava sempre com sessões de treinamento guiada por um *coach* experiente. A maioria dos projetos era composto por times sem experiência nas práticas propostas. A experiência de 5 anos do autor com Métodos Ágeis e XP mostrou que um número reduzido de pessoas nos níveis 2 e 3 não afetou a adoção de um Método Ágil quando a equipe é guiada por um bom *coach*. A partir dessa experiência, notou-se que, em geral, é mais fácil ensinar Métodos Ágeis para programadores inexperientes. Programadores mais experientes geralmente tendem a demonstrar maior resistência à práticas ágeis como Desenvolvimento Dirigido por Testes, Código Compartilhado e Programação Pareada. Essa resistência acontece pois eles precisam mudar drasticamente seu modo de trabalho, alterando algo que já estão acostumados há diversos anos.

Além disso, percebeu-se que existe outro fator pessoal associado ao sucesso na adoção de

um Método Ágil: a experiência e influência do *coach*. No Projeto 6, conforme o nível de participação dos consultores foi diminuindo, o papel do *coach* foi passado aos funcionários do governo que trabalhavam em tempo integral. Como sua experiência com XP e influência na equipe era menor, algumas práticas começaram a ser deixadas de lado, conforme será discutido na Seção 4.3.5.

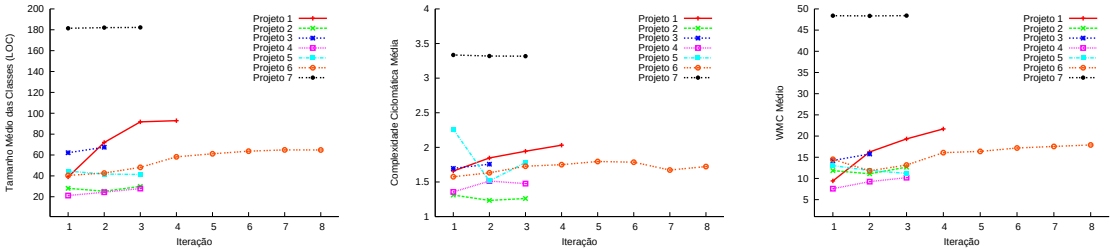
Dessa forma, existem outros fatores pessoais que influenciam a escolha entre um Método Ágil e um método com mais ênfase no planejamento (*plan-driven*) além da experiência da equipe com processos adaptativos. É preciso conduzir estudos mais minuciosos para entender como as mudanças culturais trazidas pelos Métodos Ágeis podem influenciar as pessoas de uma equipe e para descobrir quais os fatores de risco relevantes que influenciam a adoção de um Método Ágil.

4.3.4. Métricas de Acompanhamento para Design e Qualidade do Código

Chidamber e Kemerer propuseram uma família de métricas para avaliar a qualidade de sistemas orientado a objetos, conhecida como *suite CK* [30], que foi amplamente validada na literatura [5,19]. Segundo Chidamber e Kemerer, essas métricas auxiliam os desenvolvedores a entender a complexidade do design, detectar falhas e prever alguns atributos de qualidade do sistema produzido como quantidade de defeitos e esforço para teste e manutenção. Num estudo recente [96], o autor avaliou um subconjunto dessa família e os resultados são apresentados nesta seção.

Tamanho e Métricas de Complexidade: LOC, $v(G)$ e WMC

Os valores médios para LOC, $v(G)$ e WMC em cada iteração são apresentados nas Figuras 4.3(a), 4.3(b) e 4.3(c), respectivamente. O formato dos três gráficos apresentam uma evolução parecida. Na verdade, o valor da correlação por *rank* de Spearman entre essas métricas (vide Tabela 4.12) mostram que elas são altamente dependentes.



(a) LOC (b) $v(G)$ (c) WMC

Figura 4.3.: Evolução dos valores médios de LOC, $v(G)$ e WMC

Métricas	**Correlação (ρ)**	**p-value**
LOC vs. $v(G)$	0.861	< 0.000001
LOC vs. WMC	0.936	< 0.000001
$v(G)$ vs. WMC	0.774	< 0.00001

Tabela 4.12.: Resultados dos testes de correlação por *rank* de Spearman

O Projeto 7 possui uma média significativamente maior de LOC, $v(G)$ e WMC em relação aos outros projetos. Este foi o projeto onde apenas algumas das práticas ágeis foram adotadas. Na verdade, ele possui a implementação mais deficiente de XP, conforme discutido na Seção 4.2.6. Isso sugere que o Projeto 7 vai estar mais sujeito a apresentar defeitos e erros, exigindo maior esforço de teste e manutenção. Ao comparar o Projeto 7 com dados obtidos na literatura, percebe-se que projetos com LOC média parecida (183.27 [51] e 135.95 [104]) apresentam um WMC médio significativamente inferior (17.36 [51] e 12.15 [104]). Outros estudos mostram um WMC médio parecido, porém o tamanho não foi controlado: 13.40 [5], 11.85, 6.81 e 10.37 [105]. Esses valores de WMC são mais consistentes com os outros seis projetos ágeis, porém os projetos ágeis apresentam classes menores (LOC menor).

Também é possível perceber uma tendência de crescimento com o passar das iterações. Essa tendência é mais acentuada nas iterações iniciais de projetos novos (como o Projeto 1), confirmando o resultado de Alshayeb e Li [1]. Após algumas iterações a tendência de crescimento tende a se estabilizar. A única exceção é o Projeto 5, que mostra uma redução no tamanho e

na complexidade. Isso pode ser explicado pela falta de ênfase nos testes automatizados e na refatoração durante a primeira fase do desenvolvimento. A equipe não possuia as habilidades necessárias para escrever testes automatizados para o módulo em J2ME antes das aulas da disciplina de XP. Isso sugere que as práticas de teste e refatoração são eficientes para controlar o tamanho e a complexidade das classes de um sistema e que métricas como LOC, $v(G)$ e WMC são eficientes para o acompanhamento de tais práticas.

Métrica de Coesão: LCOM

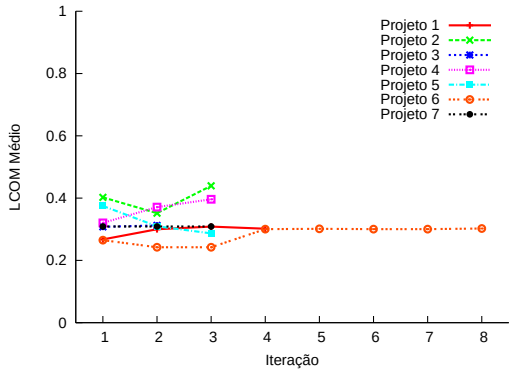
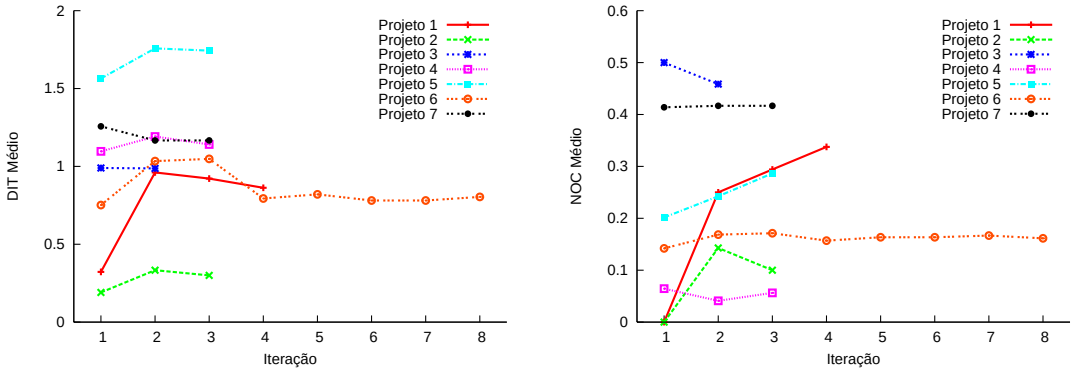


Figura 4.4.: Evolução dos valores médios de LCOM

O valor médio para LCOM em cada iteração é apresentado na Figura 4.3.4. Entretanto, nenhum resultado interessante pôde ser obtido dessa métrica devido à similaridade de valores entre os projetos. Na verdade, a relação dessa métrica com a qualidade do código é controversa na literatura: enquanto Basili et al. mostraram que LCOM era insignificante [5], Gyimóthy et al. encontraram uma relação significativa [51].

Métricas de Herança: DIT e NOC

Os valores médios para DIT e NOC em cada iteração são apresentados nas Figuras 4.5(a) e 4.5(b), respectivamente. O uso dessas métricas para prever a possibilidade de defeitos nas classes de um sistema também é controverso na literatura [29, 51]. A Tabela 4.13 mostra os



(a) DIT

(b) NOC

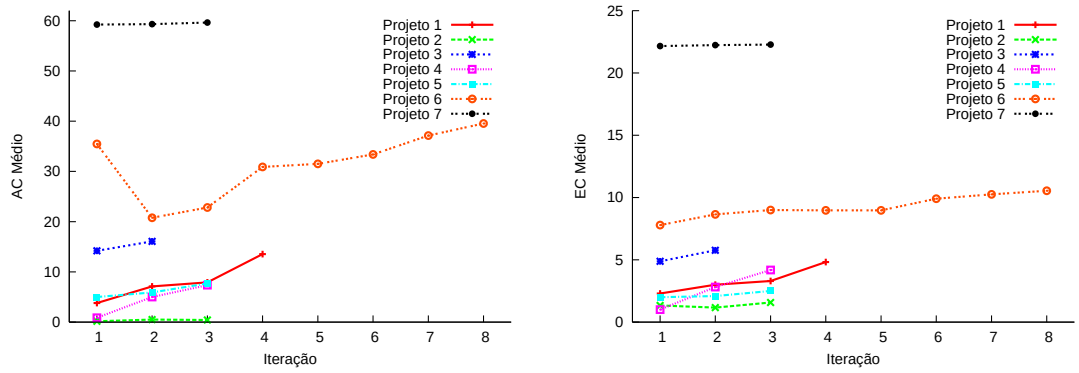
Figura 4.5.: Evolução dos valores médios de DIT e NOC

valores médios para DIT e NOC em diversos estudos, para comparação.

Métrica	[5]	[51]	[105] A	[105] B	[105] C	[104]	[29]
DIT	1.32	3.13	1.25	1.54	0.89	1.02	0.44
NOC	0.23	0.92	0.20	0.70	0.24	N/A	0.31

Tabela 4.13.: Valores médios de DIT e NOC na literatura

Nenhum dos projetos apresentados neste capítulo apresentam valores altos para DIT e NOC, mostrando que não houve abuso no uso de herança. Valores médios de DIT próximos de 1.0 podem ser explicados pelo uso de arcabouços como Struts e Swing, que oferecem funcionalidades através da extensão de suas classes base. Em particular, grande parte do código do Projeto 5 representa uma aplicação móvel e algumas de suas classes base herdam diretamente das classes de interface gráfica do J2ME, resultando em valores de DIT maiores. NOC é geralmente menor para projetos novos e uma tendência de crescimento pode ser observada em alguns projetos. Isso pode ser explicado pelo fato de que grande parte da evolução de um sistema envolve a adaptação e extensão de comportamentos existentes.



(a) AC

(b) EC

Figura 4.6.: Evolução dos valores médios de AC e EC

Métricas de Acoplamento: AC e EC

Os valores médios para AC e EC em cada iteração são apresentados nas Figuras 4.6(a) e 4.6(b), respectivamente. O formato desses dois gráficos mostra uma evolução parecida. Na verdade, existe uma alta dependência entre essas duas métricas. Foi encontrada uma correlação por *rank* de Spearman de 0.971 com relevância estatística num nível de confiança de 95% (p-value $< 10^{-14}$). Porém, infelizmente, não é possível comparar esses resultados com outros estudos devido à diferenças na forma em que as métricas de acoplamento foram calculadas. A métrica mais comum na literatura é o CBO, que é parecido com EC porém calculado no escopo de uma classe, e não de um pacote, como foi calculado pelo plug-in Eclipse Metrics, descrito na Seção 4.1.

Mais uma vez, o Projeto 7 apresenta valores médios de AC e EC superiores aos outros projetos. Binkley e Schach mostraram que métricas de acoplamento são bons indicadores para avaliar esforços de manutenção [19]. Nesse caso, como o desenvolvimento do sistema foi terceirizado, o time já estava passando por dificuldades com sua manutenção. Além disso, não havia testes automatizados para servir como rede de segurança ao fazer alterações no código. Foi obtido algum sucesso na adoção de Integração Contínua, conforme será descrito

na Seção 4.3.5, porém a adoção de testes de unidade automatizados não foi tão bem recebida pela equipe. Escrever testes de unidade para um sistema com muito código legado é bem mais difícil e exige bastante habilidade técnica [43]. Apesar disso, a adoção de testes de aceitação automatizados foi um pouco melhor recebida pela equipe, através do uso de ferramentas como o Selenium[2] e o Selenium IDE[3].

4.3.5. Métrica de Acompanhamento para Integração Contínua

A Integração Contínua é uma das práticas mais importantes de XP que permite a entrega e implantação de software funcionando ao final de cada iteração ou *release* [44]. Ela é uma técnica que reduz o risco de grandes integrações ao final dos ciclos de desenvolvimento ao disponibilizar:

- O *build* automatizado do sistema completo;
- Execução freqüente da bateria completa de testes;
- Meios para o Time Completo entender o que está acontecendo com o sistema.

Para acompanhar a adoção da Integração Contínua, foi utilizado o Fator de Integração IF_i, apresentado na Seção 3.5.3. A Figura 4.7 mostra o valor do fator de integração para cada iteração de todos os projetos:

Novamente é possível observar que os membros do Projeto 7 estavam acostumados a esperar mais antes de fazer o *commit* de suas alterações no repositório. Este comportamento começou a mudar após a introdução da prática da Integração Contínua à equipe, o que pode ser observado por uma queda brusca na linha do gráfico. Também é possível observar uma tendência crescente no fator de integração do Projeto 6. Como a equipe de consultores da USP foi gradualmente deixando a liderança do projeto nas mãos dos funcionários da ALESP,

[2] http://www.openqa.org/selenium
[3] http://www.openqa.org/selenium-ide

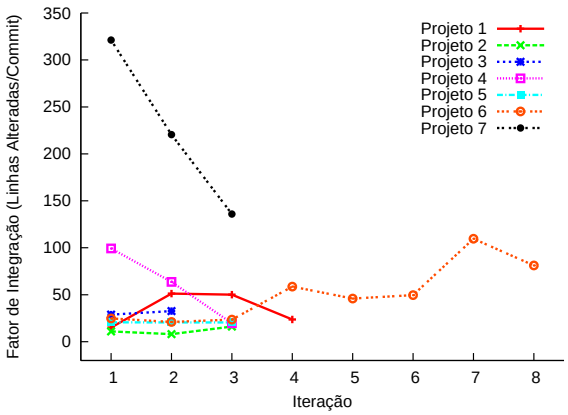


Figura 4.7.: Fator de Integração

eles mantiveram algumas práticas porém passaram a ser menos rigorosos com outras práticas como a Integração Contínua. Isso mostra que as mudanças propostas por algumas práticas ágeis são difíceis de ser sustentadas, exigindo atenção e disciplina constante da equipe. Ao contrário do que o senso comum pode sugerir, adotar e, principalmente, manter um Método Ágil como XP requer muita disciplina.

Para avaliar se essa métrica é apropriada, foi calculada a correlação por *rank* de Spearman, comparando o Fator de Integração com a avaliação média da prática de Integração Contínua ao final da última iteração de cada equipe. Uma correlação negativa de -0.57 foi encontrada, porém sem relevância estatística num nível de confiança de 95% (p-value $= 0.1, N = 7$). Isso provavelmente aconteceu devido ao tamanho reduzido da amostra, uma vez que o questionário foi respondido apenas uma vez no final do semestre (após a última iteração de cada projeto).

4.3.6. Métrica de Acompanhamento para Testes e Qualidade

Conforme discutido na Seção 2.9.4, a qualidade é um dos princípios mais importantes da Programação Extrema. Beck e Fowler afirmam que o planejamento em XP nem considera a qualidade como uma variável de controle do projeto [16]. Diversas das práticas de XP enfatizam a preocupação constante com a qualidade do sistema produzido, como o Desen-

volvimento Dirigido por Testes , a Integração Contínua, o Design Incremental e a Refatoração. Em particular, XP exige que tanto os testes de unidade quanto os testes de aceitação sejam automatizados, para que sejam executados continuamente, fornecendo *feedback* sobre a qualidade.

Para acompanhar a adoção das práticas de teste, foi utilizado o Fator de Teste T_i, apresentado na Seção 3.5.3. A Figura 4.8 mostra o valor do fator de teste para cada iteração de todos os projetos:

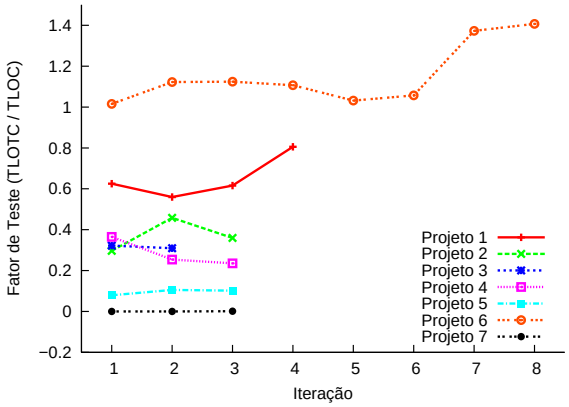


Figura 4.8.: Fator de Teste

Observa-se que alguns projetos apresentam mais linhas de código de teste do que linhas de código de produção ($T_i > 1$). É possível também notar iterações em que os testes foram deixados de lado, causando uma queda no Fator de Teste e demonstrando que mais código de produção foi escrito sem a cobertura dos testes automatizados. Além disso, observa-se um baixo Fator de Teste para os Projetos 5 e 7 nos quais o desenvolvimento começou sem a utilização de um Método Ágil. O formato plano do gráfico para esses projetos sugere a dificuldade de testar sistemas legados, onde grande parte do código já foi desenvolvida sem a cobertura dos testes automatizados. Apesar dos desenvolvedores começarem a escrever testes para o código legado, os sistemas tinham aproximadamente 10.000 e 170.000 linhas de código, respectivamente. Um grande esforço ainda seria necessário para melhorar a cobertura dos

testes. Testes automatizados são ferramentas muito eficientes para a construção e manutenção de sistemas de software com alta qualidade, por isso é importante desenvolver uma cultura de teste na equipe desde o início do projeto.

Para avaliar se essa métrica é apropriada, foi calculada a correlação por *rank* de Spearman, comparando o Fator de Teste com a avaliação média da prática de teste ao final da última iteração de cada equipe. Uma correlação positiva de 0.72 foi encontrada com relevância estatística num nível de confiança de 95% (p-value $= 0.03382, N = 7$).

4.4. Discussão e Conclusões

Este capítulo apresentou um estudo de caso empírico no qual foram analisados sete projetos acadêmicos e governamentais sob o ponto de vista do acompanhamento de um processo ágil. Os projetos foram conduzidos por *coaches* experientes em equipes compostas por membros sem prévia experiência com Métodos Ágeis. Os projetos foram classificados de acordo com o *Extreme Programming Evaluation Framework* (XP-EF), contribuindo para a construção de uma base empírica mais sólida de dados sobre XP e Métodos Ágeis.

Além disso, foram coletadas diversas métricas de acompanhamento que foram analisadas para entender sua aplicabilidade no auxílio ao gerenciamento e acompanhamento dos projetos ágeis. Devido a diversidade das métricas coletadas e da possibilidade ainda maior de possíveis medições, o *tracker* de uma equipe ágil deve escolher muito bem quais métricas utilizar. A tarefa de relacionar as métricas com as dificuldades em práticas específicas de XP não é fácil.

Alguns dos resultados obtidos a partir da análise das métricas avaliadas neste estudo de caso foram:

- Todos os projetos apresentaram uma nota desejada alta para cada prática analisada, mostrando uma vontade de melhorar a adoção das práticas de XP. A diferença entre as notas desejadas e atuais podem indicar quais os principais pontos de melhoria.

- As Retrospectivas e seus cartazes são ferramentas complementares ao acompanhamento, ajudando o time a entender o andamento do projeto. Mesmo as equipes que não estavam indo bem com o *tracking* puderam se beneficiar das reuniões de Retrospectivas.

- Um número baixo de pessoas nos níveis 2 e 3, com capacidade para adaptar o método para situações inesperada, não afetou o sucesso na adoção de um Método Ágil como XP. Existem outros fatores pessoais que podem influenciar a adoção de um Método Ágil, como a experiência e a capacidade de influência do *coach*. Novos estudos devem ser conduzidos para investigar como as mudanças culturais trazidas pelos Métodos Ágeis podem influenciar as pessoas de uma equipe e para descobrir quais os fatores de risco são relevantes para escolher entre uma abordagem ágil ou com mais ênfase no planejamento (*plan-driven*).

- Ao analisar os resultados apresentados neste capítulo com outros da literatura, percebeu-se que o projeto que utilizava menos práticas ágeis (Projeto 7) apresenta métricas de tamanho, complexidade e acoplamento maiores (LOC, $v(G)$, WMC, AC e EC). Isso sugere que ele estará mais sujeito a defeitos e precisará de maior esforço de testes e manutenção. Também percebeu-se uma alta correlação entre as métricas de tamanho e complexidade (LOC, $v(G)$ e WMC) e entre as métricas de acoplamento (AC e EC).

- Uma métrica de acompanhamento para Integração Contínua foi analisada. A correlação por *rank* de Spearman entre o Fator de Integração e a avaliação da equipe da prática de Integração Contínua foi calculada como -0.57, porém sem relevância estatística num nível de confiança de 95% (p-value $= 0.1, N = 7$) devido à pequena amostra analisada.

- Uma métrica de acompanhamento para testes e qualidade foi analisada. A correlação por *rank* de Spearman entre o Fator de Teste e a avaliação da equipe da prática de testes foi calculada como 0.72, com relevância estatística num nível de confiança de 95% (p-value $= 0.03382, N = 7$).

5. Conclusão

Este trabalho estudou o papel das métricas no acompanhamento (*tracking*) de projetos utilizando Métodos Ágeis de desenvolvimento de software. Além da discussão sobre os conceitos e teorias relacionados aos Métodos Ágeis e às métricas, um estudo de caso foi conduzido para validar a aplicação de algumas métricas em projetos de diferentes contextos.

Apresentamos um pouco da história da engenharia de software, enfatizando as evidências que levaram ao surgimento dos Métodos Ágeis de desenvolvimento de software. Diferentes abordagens e práticas já vinham sendo utilizadas na indústria, até que alguns dos líderes se juntaram para escrever o Manifesto Ágil. Com base nos seus valores e princípios, diversas metodologias surgiram, em particular a Programação Extrema (XP) foi um dos Métodos Ágeis que mais recebeu atenção na virada do século.

Os valores, princípios e práticas de XP foram apresentados e discutidos, traçando um paralelo entre o que foi proposto na primeira e na segunda versão. Devido a natureza empírica do processo proposto por XP, este trabalho apresentou também algumas formas de adaptação da metodologia, através da utilização de práticas originadas em outros Métodos Ágeis como Scrum, Crystal e Lean.

Com base nesse ciclo constante de inspeção, adaptação e melhoria, foram apresentados os conceitos e classificações das diferentes métricas que podem ser utilizadas para acompanhamento do andamento de um projeto ágil. Em particular, foi discutida a diferença entre a utilização de métricas organizacionais, coletadas num nível mais amplo para evitar a micro-otimização, e métricas de acompanhamento, coletadas localmente por cada equipe para

avaliação, inspeção e adaptação do processo. Uma das principais diferenças entre a utilização de métricas organizacionais ou métricas de acompanhamento está no caráter de avaliação de desempenho. Métricas de acompanhamento devem ser projetadas de forma a ocultar o desempenho individual, enquanto métricas organizacionais avaliam o desempenho do todo. A avaliação de desempenho é um dos principais incentivos que levam as pessoas a se comportarem de forma a otimizar a métrica ao invés do processo.

Este trabalho discutiu também as diferentes abordagens para escolha das métricas mais apropriadas, sugerindo uma abordagem mista de acordo com as necessidades do *tracker* e da equipe. O modelo Objetivo-Pergunta-Métrica (*Goal Question Metric* ou GQM) propõe uma abordagem *top-down* para a definição de métricas, partindo de um objetivo claro para chegar às métricas apropriadas. Isso permite a discussão dos incentivos gerados por cada métrica e evita a coleta das medidas inapropriadas, porém sua estrutura hierárquica estimula a proliferação de diversas métricas. Já a abordagem Lean sugere um menor número de métricas organizacionais, através de medições em um nível acima. Isso evita a proliferação de métricas e gera incentivos para que os responsáveis pelo fluxo de entrega de valor do sistema colaborem eficientemente. A combinação de ambas as abordagens funciona bem para a definição e escolha das métricas organizacionais, porém não fornece uma direção precisa para escolha das métricas de acompanhamento. Para isso, este trabalho propõe as reuniões de Retrospectiva, que fomentam a discussão dos pontos de melhoria do processo entre os membros da equipe, auxiliando na escolha das melhores métricas de acompanhamento.

Através de um estudo de caso, este trabalho avaliou o papel das métricas de acompanhamento em sete projetos de diferentes contextos, cinco realizados num ambiente acadêmico e dois realizados num ambiente governamental. Os sete projetos fora classificados nos termos e categorias propostos pelo *Extreme Programming Evaluation Framework* (XP-EF), um arcabouço que descreve o contexto do estudo de caso, a extensão da adoção das práticas de XP e os resultados dessa adoção.

Por fim, este estudo de caso validou o uso das métricas de acompanhamento. A importância das reuniões de Retrospectiva no acompanhamento do projeto foi confirmada, assim como a relevância de algumas métricas objetivas relacionadas ao tamanho e complexidade do código (LOC, $v(G)$ e WMC), acoplamento entre classes e módulos do sistema (AC e EC), testes e qualidade (Fator de Teste) e Integração Contínua (Fator de Integração).

A escolha das melhores métricas em uma determinada situação vai depender do contexto de cada equipe. Assim como o processo é adaptativo, as métricas escolhidas também devem o ser. Uma métrica de acompanhamento deve ser utilizada somente enquanto estiver trazendo informações relevantes, devendo ser descartada assim que deixar de ser útil. As reuniões de Retrospectiva são um bom momento para discutir se as métricas devem ser adaptadas e se o seu uso está trazendo valor para a melhoria do processo.

5.1. Contribuições

Dentre as principais contribuições deste trabalho, destacam-se:

- A investigação do papel das métricas em projetos que utilizam Métodos Ágeis de desenvolvimento de software, através da definição de conceitos, formas de classificação e discussão das diferentes abordagens para escolha das métricas em uma situação específica.

- A apresentação de diversos exemplos de medidas e diferentes tipos de métricas, algumas delas validadas num estudo de caso com sete projetos.

- A classificação dos sete projetos no XP-EF, contribuindo para a construção de uma base de evidências empíricas da adoção dos Métodos Ágeis e da Programação Extrema.

- Um questionário adaptado de Krebs [70] para avaliação da aderência as práticas de XP, incluindo uma discussão dos resultados obtidos nos sete projetos analisados.

- Um catálogo de métricas para o *tracker* de uma equipe XP, com exemplos de medidas e métricas que podem ser utilizadas para acompanhamento de práticas específicas de XP.

5.2. Trabalhos Futuros

O estudo de caso apresentado neste trabalho avaliou apenas o uso das métricas de acompanhamento. Seria interessante conduzir mais estudos para validação de outras métricas de acompanhamento e, principalmente, métricas organizacionais. Além disso, na análise do estudo apresentado, dentre os fatores de risco propostos por Boehm e Turner para escolher entre uma abordagem ágil ou com mais ênfase no planejamento (*plan-driven*) [24, 25], os fatores pessoais que podem influenciar a adoção de um Método Ágil como XP ficaram em aberto. Um número baixo de pessoas nos níveis 2 e 3 propostos por Cockburn [34], com capacidade para adaptar o método para situações inesperadas, não afetou o sucesso na adoção de XP. Existem outros fatores pessoais que podem influenciar a adoção de um Método Ágil, como a experiência e a capacidade de influência do *coach*. Novos estudos devem ser conduzidos para investigar como as mudanças culturais trazidas pelos Métodos Ágeis podem influenciar tais fatores pessoais.

Além disso, com base nas respostas ao questionário de aderência a XP e na coleta de novos dados relacionados a defeitos e *bugs*, seria interessante fazer uma análise multivariada para descobrir se existe correlação entre a aderência às práticas de XP e a qualidade final do software produzido, sob o ponto de vista do usuário final.

Por fim, o questionário e o catálogo de métricas apresentados neste trabalho poderiam ser adaptados para considerar as novas práticas de XP, assim como outras práticas adaptadas de outras metodologias.

Referências Bibliográficas

[1] Mohammad Alshayeb and Wei Li. An empirical validation of object-oriented metrics in two different iterative software processes. *IEEE Transactions on Software Engineering*, 29(11):1043–1049, 2003. 89

[2] Scott Ambler. Crossing the chasm. Dr. Dobb's Journal, disponível em: www.ddj.com/dept/architect/187200223, May 2006. Acessado em: 30/10/2006. 17

[3] Ken Auer and Roy Miller. *Extreme Programming Applied: Playing to Win*. Addison-Wesley Professional, 2001. 37

[4] Robert D. Austin. *Measuring and Managing Performance in Organizations*. Dorset House Publishing Company, Inc., 1996. 45

[5] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996. 88, 89, 90, 91

[6] Vitor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, pages 528–532, 1996. 43, 44

[7] Kent Beck. SUnit project. Disponível em: sunit.sourceforge.net/. Acessado em: 30/10/2006. 14

[8] Kent Beck. Simple smalltalk testing: With patterns. Technical report, First Class Software, Inc., Outubro 1994. Disponível em: www.xprogramming.com/testfram.htm. 14

[9] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999. 3, 4, 14, 30, 33

[10] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, 2003. 15, 17, 27, 121

[11] Kent Beck and Cynthia Andres. Getting started with XP: Toe dipping, racing dives, and cannonballs. Disponível em: www.threeriversinstitute.org/ToeDipping.pdf. Acessado em: 30/10/2006. 35

[12] Kent Beck and Cynthia Andres. *Extreme Programming Explained : Embrace Change*. Addison-Wesley Professional, 2nd edition, 2004. 3, 4, 13, 15, 17, 18, 20, 23, 28, 33, 35, 54, 59, 120

[13] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. Disponível em: www.agilemanifesto.org, 2001. Acessado em: 30/10/2006. 3, 9, 16, 17

[14] Kent Beck and Ward Cunningham. Using pattern languages for object-oriented programs. Technical Report CR–87–43, Tektronix, Inc., September 1987. Presented at the OOPSLA-87 Workshop on Specification and Design for Object-Oriented Programming. 14

[15] Kent Beck and Ward Cunningham. A laboratory for teaching object-oriented thinking. In *OOPSLA*, pages 1–6, October 1989. 14

[16] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Professional, 2000. 22, 51, 59, 94

[17] Kent Beck and Erich Gamma. JUnit project. Disponível em: www.junit.org/. Acessado em: 30/10/2006. 14

[18] Kent Beck and Erich Gamma. Test infected: Programmers love writing tests. *Java Report*, 3(7), July 1998. Acessado em Jan/06. 14

[19] Aaron B. Binkley and Stephen R. Schach. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In *20th International Conference on Software Engineering*, pages 452–455, 1998. 88, 92

[20] Barry W. Boehm. A spiral model of software development and enhancement. *SIGSOFT Software Engineering Notes*, 11(4):14–24, 1986. 2

[21] Barry W. Boehm. Industrial software metrics top 10 list. *IEEE Software*, 4(5):84–85, Set 1987. 8

[22] Barry W. Boehm and Victor R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, Jan 2001. 8

[23] Barry W. Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, Oct 1988. 8

[24] Barry W. Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, 2003. 53, 69, 86, 102

[25] Barry W. Boehm and Richard Turner. Using risk to balance agile and plan-driven methods. In *IEEE Computer*, volume 36, pages 57–66, 2003. 86, 102

[26] Laurent Bossavit. The unbearable lightness of programming: a tale of two cultures. Whitepaper, disponível em: www.exoftware.com/i/white_paper/file/6/The_20unbearable_20.pdf. Acessado em: 30/10/2006. 36

[27] Piergiuliano Bossi. eXtreme Programming applied: a case in the private banking domain. Disponível em: www.quinary.com/pagine/downloads/files/Resources/OOP2003Paper.pdf, 2003. Acessado em: 30/10/2006. 4

[28] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Professional, anniversary edition, 1995. 2

[29] Michelle Cartwright and Martin Shepperd. An empirical investigation of an object-oriented software system. *IEEE Transactions on Software Engineering*, 26(7):786–796, 2000. 90, 91

[30] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. 52, 88, 123

[31] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. *CMMI : Guidelines for Process Integration and Product Improvement*. Addison-Wesley Professional, 2003. 2

[32] Peter Coad, Jeff de Luca, and Eric Lefebvre. *Java Modeling Color with UML: Enterprise Components and Process with Cdrom*. Prentice Hall, 1999. 12

[33] Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 2004. 11, 33

[34] Alistair Cockburn. *Agile Software Development: The Cooperative Game*. Addison Wesley Professional, 2nd edition, 2006. 3, 24, 37, 46, 53, 86, 87, 102

[35] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall PTR, 2005. 25, 39

[36] Jim Collins. *Good to Great: Why Some Companies Make the Leap... and Others Don't*. Collins, 2001. 42

[37] Ward Cunningham. System of names. Disponível em: c2.com/cgi/wiki?SystemOfNames. Acessado em: 30/05/2007. 31

[38] Dennis de Champeaux. Software engineering considered harmful. *Communications of the ACM*, 45(11):102–104, 2002. 2

[39] Tom Demarco and Timothy Lister. *Peopleware: Productive Projects and Teams*. Dorset House Publishing Company, Incorporated, 2nd edition, 1999. 24

[40] W. Edwards Deming. *Out of The Crisis*. Massachusetts Institute of Technology, 1986. 46

[41] Yael Dubinsky, David Talby, Orit Hazzan, and Arie Keren. Agile metrics at the israeli air force. In *Agile 2005 Conference*, pages 12–19, 2005. 50

[42] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003. 31

[43] Michael Feathers. *Working Effectively With Legacy Code*. Prentice Hall PTR, 2004. 93

[44] Martin Fowler. Continuous integration. Disponível em: www.martinfowler.com/articles/continuousIntegration.html, July 2006. Acessado em: 30/10/2006. 79, 93

[45] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999. 14, 15, 30, 79

[46] Alexandre Freire da Silva, Fabio Kon, and Cicero Torteli. XP south of the equator: An experience implementing XP in brazil. In *Proceedings of the 6th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP'2005)*, pages 10–18, 2005. 4

[47] Bernard A. Galler. ACM president's letter: NATO and software engineering? *Communications of the ACM*, 12(6):301, 1969. 1

[48] Jane F. Gilgun. Definitions, methodologies, and methods in qualitative family research. *Qualitative Methods in Family Research*, 1992. 41

[49] Alfredo Goldman, Fabio Kon, Paulo J. S. Silva, and Joe Yoder. Being extreme in the classroom: Experiences teaching XP. *Journal of the Brazilian Computer Society*, 10(2):1–17, 2004. 68

[50] Eliyahu M. Goldratt. *The Haystack Syndrome: Sifting Information Out of the Data Ocean*. North River Press, 1991. 37

[51] Tibor Gyimóthy, Rudolf Ferenc, and István Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005. 89, 90, 91

[52] Thomas Haigh. Software in the 1960s as concepts, service, and product. In *IEEE Annals of The History of Computing*, volume 24, pages 5–13, 2002. 1

[53] Deborah Hartmann and Robin Dymond. Appropriate agile measurements: Using metrics and diagnostics to deliver business value. In *Agile 2006 Conference*, pages 126–131, 2006. 42, 47, 49, 54, 59

[54] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall PTR, 1996. 52

[55] Jim Highsmith. Messy, exciting, and anxiety-ridden: Adaptive software development. In *American Programmer*, volume 10, 1997. 12

[56] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122, 2001. 2

[57] IEEE standard glossary of software engineering terminology, 1990. IEEE Std 610.12. 38

[58] IEEE standard glossary of software engineering terminology, 1983. IEEE Std 729. 38

[59] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 2

[60] Ron Jeffries. A metric leading to agility. Disponível em: www.xprogramming.com/xpmag/jatRtsMetric.htm, 2004. Acessado em: 31/05/2007. 54

[61] Ron Jeffries, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed*. Addison-Wesley Professional, 2000. 16, 37

[62] Kay Johansen, Ron Stauffer, and Dan Turner. Learning by doing: Why XP doesn't sell. In Proceedings of the XP/Agile Universe, 2001. 3

[63] Jim Johnson. ROI, it's your job. Keynote Speech at Third International Conference on Extreme Programming (XP2002), May 2002. 8

[64] Luanne (James) Johnson. A view from the 1960s: How the software industry began. *IEEE Annals of the History of Computing*, 20(1):36–42, 1998. 1

[65] Jim Johnson, et al. CHAOS in the new millenium. Technical report, Standish Group, 2000. 8

[66] Capers Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison Wesley Professional, 2000. 68

[67] Mira Kajko-Mattsson, Ulf Westblom, Stefan Forssander, Gunnar Andersson, Mats Medin, Sari Ebarasi, Tord Fahlgren, Sven-Erik Johansson, Stefan Törnquist, and Margareta Holmgren. Taxonomy of problem management activities. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 1–10, 2001. 7

[68] Raymond Kehoe and Alka Jarvis. *ISO 9000-3: A Tool for Software Product and Process Improvement*. Springer, 1995. 2

[69] Norman L. Kerth. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House Publishing Company, 2001. 33

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

[71] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2003. 1, 3, 9, 16

[72] Bo Leuf and Ward Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, 2001. 14

[73] Kim Man Lui and Keith C.C. Chan. Test driven development and software process improvement in china. In *Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP'2004)*, volume 3092 of *Lecture Notes on Computer Science*, pages 219–222, 2004. 4

[74] Chris Mann and Frank Maurer. A case study on the impact of Scrum on overtime and customer satisfaction. In *Agile 2005 Conference*, pages 70–79, 2005. 4

[75] Michele Marchesi. The new XP. Disponível em: www.agilexp.org/downloads/TheNewXP.pdf. Acessado em: 30/05/2007. 31, 32

[76] Thomas J. McCabe and Arthur H. Watson. Software complexity. *Crosstalk: Journal of Defense Software Engineering*, 7:5–9, 1994. 51

[77] John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, and Fred Hall. *Practical Software Measurement: Objective Information for Decision Makers*. Addison-Wesley Professional, 2002. 38

[78] Jacques Morel et al. XPlanner website. Disponível em: www.xplanner.org. Acessado em: 30/10/2006. 66

[79] Roger A. Müller. Extreme programming in a university project. In *Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP'2004)*, volume 3092 of *Lecture Notes on Computer Science*, pages 312–315, 2004. 4

[80] Peter Naur and Brian Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, October 1968. 1

[81] Niko-niko calendar website. Disponível em: www.geocities.jp/nikonikocalendar/index_en.html Acessado em: 30/10/2006. 67, 69

[82] Taiichi Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1998. 11, 28

[83] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois, 1992. 14

[84] Stephen R. Palmer and John M. Felsing. *A Practical Guide to Feature Driven Development*. Prentice Hall, 2002. 12

[85] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit for Software Development Managers*. Addison-Wesley Professional, 2003. 3, 8, 11, 28, 43, 45, 54, 56

[86] Mary Poppendieck and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006. 8, 11, 28, 37, 45, 46, 54, 56, 120, 121

[87] Lawrence H. Putnam and Ware Meyers. *Measures For Excellence: Reliable Software On Time, Within Budget*. Yourdon Press Computing Series, 1992. 66, 67

[88] Brian Randell and John N. Buxton, editors. *Software Enfineering Techniques: Report of a conference sponsored by the NATO Science Committee*, October 1969. 1

[89] Fred Reichheld. *The Ultimate Question: Driving Good Profits and True Growth*. Harvard Business School Press, 2006. 54, 58

[90] Hugh Robinson and Helen Sharp. XP culture: Why the twelve practices both are and are not the most significant thing. In *Agile Development Conference*, pages 12–21. IEEE Computer Society, June 2003. 3

[91] Hugh Robinson and Helen Sharp. The characteristics of XP teams. In *5th International Conference on Extreme Programming and Agile Processes in Software Enfineering*, pages 139–147, 2004. 3

[92] Winston W. Royce. Managing the development of large software systems. In *Proceedings of IEEE Wescon*, pages 1–9, Aug 1970. 1

[93] Página do centro de saúde-escola "Samuel Pessoa" da Faculdade de Medicina da USP. Disponível em: www.fm.usp.br/cseb. Acessado em: 30/10/2006. 77

[94] Danilo Sato, Dairton Bassi, Mariana Bravo, Alfredo Goldman, and Fabio Kon. Experiences tracking agile projects: an empirical study. *Journal of the Brazilian Computer Society, Special Issue on Experimental Software Engineering*, 12(3):45–64, 2006. 4

[95] Danilo Sato, Dairton Bassi, and Alfredo Goldman. Extending extreme programming with practices from other methodologies. In *A ser publicado no Workshop de Desenvolvimento Rápido de Aplicações (WDRA'07)*, 2007. 32

[96] Danilo Sato, Alfredo Goldman, and Fabio Kon. Tracking the evolution of object oriented quality metrics. In *Proceedings of the 8th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2007)*, pages 84–92, 2007. 88

[97] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004. 11, 24, 32

[98] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001. 11, 24, 32, 59

[99] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999. 41, 67

[100] Frederick T. Sheldon, Krishna M. Kavi, Robert C. Tausworth, James T. Yu, Ralph Brettschneider, and William W. Everett. Reliability measurement: From theory to practice. *IEEE Software*, 9(4):13–20, Jul 1992. 8

[101] The CHAOS report. Technical report, Standish Group, 1994. 2, 8

[102] The CHAOS report. Technical report, Standish Group, 2003. 2, 8

[103] Jennifer Stapleton. *DSDM: A framework for business centered development*. Addison-Wesley Professional, 1997. 12

[104] Ramanath Subramanyam and M.S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297–310, 2003. 89, 91

[105] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *6th International Software Metrics Symposium*, pages 242–249, 1999. 89, 91

[106] Vinícius Manhães Teles. *Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*. Editora Novatec, 2004. 18

[107] Christ Vriens. Certifying for CMM level 2 and ISO9001 with XP@scrum. In *Agile Development Conference*, pages 120–124. IEEE Computer Society, 2003. 11

[108] William Wake. XP radar chart. Disponível em: www.xp123.com/xplor/xp0012b/index.shtml, Jan 2001. Acessado em: 31/05/2007. 82

[109] Arthur H. Watson and Thomas J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. Technical report, NIST Special Publication 500-235, 1996. 51

[110] Don Wells. Fix XP when it breaks. Disponível em: www.extremeprogramming.org/rules/fixit.html. Acessado em 31/05/2007. 32

[111] Laurie Williams. *Pair Programming Illuminated*. Addison-Wesley Professional, 2002. 70

[112] Laurie Williams, William Krebs, Lucas Layman, and Annie I. Antón. Toward a framework for evaluating extreme programming. In *8th International Conference on Empirical Assessment in Software Engineering (EASE '04)*, pages 11–20, 2004. 4, 65, 68

[113] Laurie Williams, Lucas Layman, and William Krebs. Extreme programming evaluation framework for object-oriented languages – version 1.4. Technical report, North Carolina State University Department of Computer Science, 2004. 4, 65, 68

[114] xUnit family. Disponível em: www.xprogramming.com/software.htm. Acessado em: 30/05/2007. 14

A. Questionário - Métricas de Aderência (XP-am)

O questionário utilizado no estudo de caso foi construído com base no trabalho de William Krebs sobre o assunto [70] e é apresentado a seguir. Além disso, ele também está disponível *online* em http://agilcoop.incubadora.fapesp.br/portal/Artigos/Questionario.pdf.

A.1. Questionário de Adesão – Programação Extrema (XP)

Equipe/Projeto: _____ Data: ____/____/____

A. Tempo de educação formal em Informática ou Ciência da Computação (anos)
(Para responder esta pergunta, considere a duração normal do curso e não o tempo que você demorou para concluí-lo, por exemplo, se você fez graduação em Computação e já concluiu o primeiro ano do mestrado, responda 5 anos (4 anos da graduação mais 1 de mestrado, mesmo que você tenha demorado 6 anos para concluir a graduação)

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 ( ) 6 ou mais

B. Tempo de experiência profissional em Informática (anos)

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 ( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10 ou mais

C. Número de cursos específicos em Programação Orientada a Objetos
(por exemplo: MAC 110 em Java, POO, Tópicos de POO, SOD, Padrões de Projeto, etc.)

( ) 0 ( ) 1 ( ) 2 ( ) 3 ( ) 4 ou mais

D. Responda às perguntas abaixo dando uma nota de 0 a 10 para cada uma das práticas de Programação Extrema (XP), considerando o nível atual da equipe (Atual) e o nível que considera desejável (Ideal):

Prática	Comentários
Programação Pareada Atual:_____ Desejado:_____	**Duas pessoas trabalham juntas no mesmo computador. Elas trocam de papel constantemente atuando como digitador ou revisor, pensando no trabalho que estão realizando.** 10 Nós não gostaríamos de escrever qualquer código crítico sem parear. Nós fazemos o rodízio constante de pares. 8 Nós geralmente trabalhamos em pares. 6 Nós geralmente fazemos discussões no quadro branco, conversamos no *chat* ou em visitas ao laboratório. Algumas pessoas programam em pares no teclado, mas alguns preferem não tentar. 4 Nós tentamos parear mas não conseguimos devido a desencontro de horários e reuniões. Algumas pessoas são muito rápidas ou devagares para que eu tenha paciência de sentar junto. De qualquer forma, nossos móveis tornam o pareamento difícil. 2 Eu me distraio muito quando alguém me interrompe. Meu companheiro de trabalho me pede para não ter tantos visitantes. 0 Eu uso fones de ouvido para que as pessoas não me interrompam. Na verdade, eu prefiro trabalhar em casa com o telefone fora do gancho e com o programa de *chat* configurado no modo "Ocupado".
Versões Pequenas Atual:_____ Desejado:_____	**Nós entregamos iterações pequenas e mais freqüentes ao nosso cliente.** 10 A cada 1 ou 2 semanas nosso cliente pode ter uma nova versão funcionando do nosso sistema. 8 Nós temos iterações mensais. O cliente pode escolher novas funcionalidades para a próxima versão. 6 Dentro de alguns meses nós disponibilizamos uma nova iteração para nosso cliente. 4 Nós entregamos versões beta *patches* de correção umas 4 vezes ao ano, com entregas maiores em ciclos de 8 a 12 meses. 0 Temos visão a longo prazo. A versão 1.0 do ano que vem te deixará entretido até que as verdadeiras funcionalidades sejam implementadas na versão 2.0 dentro de 18+ meses.
Integração Contínua Atual:_____ Desejado:_____	**Quando trabalhando em uma funcionalidade, sincronizo e disponibilizo código:** 10 Diversas vezes por dia. 8 Uma vez por dia. 6 Diversas vezes por semana. 4 Uma vez por semana. 2 Algumas semanas podem se passar. Só disponibilizamos código quando a tarefa está pronta. 0 Geralmente tenho problemas porque muitas mudanças acontecem entre o momento em que eu faço *checkout* e quando tento sincronizar. Tenho que tomar cuidado na sincronização pois geralmente posso esquecer algumas coisas. O *build* parece quebrar freqüentemente.

Desenvolvimento Dirigido por Testes Atual:_____ Desejado:_____	**Temos testes automatizados para cada classe de produção?** 10 Os testes automatizados são o design. Se apagar qualquer linha do código, algum teste ficará vermelho. O cliente roda testes de aceitação. 8 Após pensar no design e escrever um pouco de código, nós escrevemos o teste automatizado. 6 Nós tomamos o cuidado de escrever testes de unidade para o nosso código antes de entregá-lo para o time de teste. 4 Nós ouvimos falar de ferramentas como o JUnit, mas nunca utilizamos. 2 Nossa fase de teste formal ao final de cada ciclo demora muito mais que o planejado porque sempre aparecem diversos *bugs* e correções devem ser realizadas. 0 Nós não temos nenhum teste formal. Os clientes geralmente nos avisam se encontrarem algum problema.
Jogo do Planejamento Atual:_____ Desejado:_____	**Nós movemos histórias para dentro e fora do plano baseado nas necessidades atuais do cliente, mantendo os prazos fixos.** 10 Antes de cada iteração curta o cliente escolhe as funcionalidades mais valiosas baseado nas estimativas dos desenvolvedores. A cada manhã nós revisamos as principais histórias numa reunião em pé de 5 minutos e os desenvolvedores se voluntariam para parear e desenvolvê-las. Como sabemos que mudanças acontecem, temos vantagem competitiva porque otimizamos nosso processo para aceitar e nos adapatar a elas. 8 Às vezes nós movemos funcionalidades para dentro e para fora do plano logo após o cliente mudar de prioridade, sempre que o time de desenvolvimento termina antes da hora ou se atrasa em alguns itens. Afinal, o produto é do cliente. Eles devem ter o que quiserem. Mudanças acontecem. 6 Planos não devem mudar. Nís cumprimos nossas datas, mesmo quando planejadas com 1 ano de antecedência. Nós criamos diversos artefatos como, por exemplo, documentos de especificação. Nós tentamos deixá-los atualizados. 4 Nós tomamos o cuidado de seguir o processo "cascata". Nós não começamos a escrever código antes de termos completado e revisado os documentos de design e especificação. Não começamos a testar antes do código ser entregue. Algumas vezes nós mudamos ou perdemos a data de entrega porque o cliente mudou algum requisito. 0 Nós perdemos clientes porque dizemos que eles devem esperar pelo próximo *release*. Afinal, nós estamos ocupados tentando terminar o que pediram no ano passado.
Cliente com os Desenvolvedores Atual:_____ Desejado:_____	**Nós temos acesso ao nosso cliente e obtemos *feedback* freqüente.** 10 Nós não consideramos uma história completa até que o cliente execute seu teste de aceitação. 8 Nós freqüentemente interagimos com nossos clientes para mostrar protótipos e ver como eles querem mudá-los. 6 Nós obtemos requisitos dos clientes. 4 Os requisitos vêm de algum lugar, mas não acho que venham dos clientes. 2 Nós entregamos funcionalidades, mas nunca temos certeza se fizemos o que eles queriam. Eles provavelmente já devem ter mudado de idéia agora. 0 Nós sabemos o que é certo. Eles usarão nosso sistema gostando ou não. Será bom para eles.

Refatoração Atual:_____ Desejado:_____	**Nós re-escrevemos ou re-projetamos código com "mau cheiro" para facilitar o desenvolvimento de novas funcionalidades.** 10 Nós geralmente usamos a refatoração como ferramenta para tornar nosso design mais fexível e facilitar a implementação de mudanças. 5 Nós realizamos alguma limpeza no código de tempos em tempos. 0 Nós temos muito código antigo que já passou por diversas mudanças ruins. Nós temos medo de alterá-lo. Nós recusamos requisitos novos pois o código funciona como especificado e não pode ser alterado.
Design Simples Atual:_____ Desejado:_____	**Nós mantemos o design simples agora para que possamos alterá-lo diante das mudanças.** 10 Nós executamos refatorações freqüentes. Nós não tentamos prever quais funcionalidades podem ser utilizadas no futuro. Generalizações precoces são ignoradas. 8 O design é limpo. Ele faz somente o necessário de uma forma simples. 6 Nosso design é intuitivo, com poucos pontos de melhoria. 4 Nós projetamos uma funcionalidade inteira com orgulho. Um produto que faz tudo que as pessoas precisam. 2 Nós temos grandes peças de código inacabado que eventualmente é jogado fora. 0 Pensei em escrever esse *framework* caso alguém precise no futuro. Nunca se sabe.
Padronização de Código Atual:_____ Desejado:_____	**Você segue uma padronização no código que permite alterações em qualquer parte do sistema?** 10 Nós temos um padrão, o seguimos e treinamos novas pessoas a seguirem. Aliás, são padrões da indústria também. 8 Nós temos boa parte do padrão documentada, e as pessoas geralmente o seguem. 6 Nós fazemos a mesma coisa em alguns casos, porém outros são tratados diferentemente. Nossas chaves são colocadas no mesmo lugar, mas em trechos de tratamento de erros estão diferentes. 4 Nós temos diversos padrões e cada um segue o seu. 2 Nós não temos padrão. 0 Como você ousa me dizer o que fazer?
Propriedade Coletiva do Código Atual:_____ Desejado:_____	**Qualquer um pode alterar qualquer parte do código. Nós não precisamos esperar o especialista.** 10 Nós mudamos código em qualquer área regularmente. Você não consegue perceber pois nosso código parece o mesmo. 8 Nós mudamos código em qualquer área regularmente. 6 Nós mudamos código alheio, mas geralmente dividimos as tarefas de acordo com a especialidade de cada um. 4 Nós podemos consertar isso se for necessário. 2 Nós teremos que esperar ele voltar de férias. 0 Eu bloqueio os arquivos do meu código e mantenho-os bloqueados.
Metáfora ou Sistema de Nomes Atual:_____ Desejado:_____	**Você tem uma convenção de nomes para seus objetos?** 10 Posso dizer o que o código de qualquer programador faz apenas lendo os nomes, sem olhar os comentários. Geralmente pensamos nos mesmos nomes para as coisas. Quando estamos discutindo sobre o design, usamos uma metáfora comum. 8 Eu gosto dos nomes no sistema. Eles fazem sentido para mim. 6 Eu posso seguir a nomenclatura, com alguma ajuda dos comentários. 4 Os nomes são confusos. Você realmente precisa entender o que o código faz. Não tenho certeza qual nome dar às classes. 2 Você precisa conhecer o histórico, o nome é um fator, mas os métodos fazem coisas diferentes pois evoluíram independentemente. 0 Por que eles usaram esse nome? Não consigo entender as abreviações.

Ritmo Sustentável Atual:_____ Desejado:_____	**As pessoas trabalham num ritmo que é produtivo e sustentável no longo prazo?** 10 Nós trabalhamos num ritmo constante e confortável. Nós corremos somente quando necessário, somente em poucas ocasiões. 8 Algumas vezes estou muito ocupado, outras muito entediado. 6 Sempre que temos uma data de entrega precisamos entrar no modo "correria". 4 Já faz alguns meses que estamos pedindo comida para a janta. Parece que sempre será assim. 2 Já precisei cancelar aulas e férias mais de uma vez. 0 Não tenho tempo para preencher esse questionário.
Lições Aprendidas Atual:_____ Desejado:_____	**Você pára periodicamente para avaliar formas de melhoria?** 10 As pessoas geralmente aparecem com idéias novas. Essas idéias são implementadas. Nós compartilhamos novas técnicas com pessoas de outra equipe. Fazemos retrospectivas periodicamente. 8 Nós pensamos no que deu errado, o que deu certo e propomos mudanças. 6 Nós temos idéias mas elas parecem desaparecer e nunca são implementadas. 4 Nós repetimos os mesmos erros. 0 Nós nunca paramos. Tenho reclamações que parecem nunca serem levadas em conta. Obviamente ninguém se importa.
Tracking Atual:_____ Desejado:_____	**Existem diversos gráficos e informações espalhadas pelas paredes que nos ajudam a entender o andamento do projeto.** 10 Atualizamos as informações diariamente e descartamos os gráficos que não trazem informações relevantes. Eles nos ajudam a entender o andamento do projeto e identificar pontos de melhoria. 8 Temos alguns gráficos interessantes que são atualizados semanalmente. 6 As informações na parede são atualizadas ao final de cada *release*. 4 Os gráficos estão desatualizados e ninguém mais se importa com o que está na parede. Precisamos terminar o trabalho para entregar o sistema no prazo. 2 Não sei qual a utilidade dos gráficos. Eles parecem não estar relacionados ao meu trabalho diário. Eu acho que se fossem retirados ninguém perceberia. 0 Não temos nenhum gráfico na parede. Preferimos guardar informações importantes em arquivos e documentos que ficam armazenados em nosso repositório central. Quem tiver interesse pode atualizá-los ou consultá-los.
Nota Geral para XP Atual:_____ Desejado:_____	**Como você avalia sua implementação geral das práticas de XP?** 10 Eu sou *coach* de XP, escrevo livros e faço apresentações em conferências. 8 Somos um bom exemplo de time de XP. 6 Nós geralmente mudamos um pouco as práticas na nossa rotina diária. Nós lemos sobre XP, mas como vidas ou muito dinheiro dependem do nosso software e como temos um time muito grande, usamos métodos formais com muita documentação e reavaliações periódicas. 4 Eu não sei o que é XP, mas as perguntas desse questionário parecem intrigantes. Talvez possamos tentar algumas delas em algum momento. 2 Não tenho certeza o que é XP, mas me parece algo ruim. 0 Não acredito que o desenvolvimento de software deva estar amarrado a um "processo", ou, nós sempre fizemos negócios da mesma forma. Não vejo necessidade de mudar agora.

Observações Finais / Sugestões: _____

B. Catálogo de Métricas para o Tracker de XP

Caso o *tracker* encontre problemas relacionados às práticas de XP na sua equipe, este catálogo mostra algumas possíveis medidas e métricas que podem ser utilizadas para a melhoria do processo. Este catálogo é um trabalho em progresso, portanto essa não é uma lista exaustiva de todas as medidas e métricas possíveis.

B.1. Programação Pareada

- **Matriz de Pareamento**: Uma matriz onde os membros da equipe são dispostos nas linhas e nas colunas e as diagonais são ignoradas. Cada vez que um par trabalha junto, uma marca é feita na linha/coluna correspondente. Essa medida vai mostrar a concentração dos pares que mais trabalham juntos e vai incentivar o rodízio.
- **Total de Horas Pareadas**: Medidas como essa, assim como porcentagem das horas pareadas, podem ser obtidas de ferramentas como o XPlanner, contanto que os membros da equipe atualizem as horas gastas em cada tarefa/história na ferramenta.
- **Total de Horas Não Pareadas**: Essa medida também pode ser obtida do XPlanner e geralmente é utilizada em tarefas além da codificação, como: documentação de manuais, horas de estudo ou pesquisa. Idealmente esse valor não deve ser alto. Uma possível métrica para acompanhar esse comportamento, derivada dessa medida e do total de horas pareadas, é o **Fator de Pareamento**, calculado como a razão entre as horas pareadas e não pareadas.

B.2. Versões Pequenas

Uma medida simples para saber se a equipe está entregando valor rapidamente é a **Duração da Iteração**. Outras medidas para esse mesmo fim são o **Total de Histórias Entregues** ou **Total de Pontos Entregues**. Essas medidas, assim como a **Velocidade** da equipe, foram apresentadas na Seção 3.5.

Poppendieck sugere que uma equipe ágil deve procurar entregar software tão freqüentemente que o cliente não tem tempo de mudar de idéia [86]. Para acompanhar esse comportamento, é possível utilizar uma medida do **Número de Mudanças na Iteração**, representando o total de histórias adicionadas ou removidas da iteração.

B.3. Integração Contínua

A Seção 3.5 apresentou algumas medidas relacionadas à Integração Contínua como: **Número de Linhas Alteradas** no repositório, **Número de *Commits*** e **Fator de Integração**. Uma outra medida possível de ser obtida do repositório de código é a **Volatilidade**, que mede o total de revisões (alterações) em um determinado arquivo. Essa medida pode ser utilizada junto com o tempo entre as revisões (um arquivo alterado sete vezes no mesmo dia é mais volátil que um arquivo que possui apenas uma revisão).

Além do repositório de código, o processo de *build* automatizado também está incluso na prática da Integração Contínua. Algumas possíveis medidas e métricas para acompanhar o processo de *build* são: **Total de *Builds* por Dia**, **Total de *Builds* Quebrados** por dia ou a **Duração Máxima de *Build* Quebrado**. Acompanhar o tempo de **Duração do *Build*** também pode auxiliar a melhorar a Integração Contínua. Kent Beck sugere que um *build* não deve demorar mais que 10 minutos [12].

B.4. Desenvolvimento Dirigido por Testes

Dentre as medidas e métricas apresentadas na Seção 3.5, o **Total de Linhas de Código de Teste** e o **Fator de Teste** estão relacionados com essa prática. Além disso, uma outra métrica comumente utilizada por equipe ágeis é a **Cobertura de Código**, que representa a porcentagem do código coberta pelos testes automatizados. Uma linha é considerada coberta quando a execução de um teste automatizado passa por ela. Segundo Kent Beck, quando alguém segue a prática de Desenvolvimento Dirigido por Testes, ele acaba com 100% do código coberto [10].

Outra métrica para acompanhar o progresso nas práticas de teste é a **Porcentagem dos Testes de Aceitação Passando**. Os testes de unidade devem passar sempre, porém os testes de aceitação automatizados são definidos pelo cliente e definem os critérios para considerar uma história completa. É esperado que essa porcentagem aumente conforme a iteração progride. Por fim, uma medida de qualidade mais ampla sobre as práticas de teste é o **Número de Defeitos** encontrados em produção. Idealmente, esse número será baixo, uma vez que os Métodos Ágeis sugerem o uso dos testes automatizados para prevenir defeitos e não para encontrá-los [86].

B.5. Jogo do Planejamento

A maioria das medidas que podem ser utilizadas para acompanhar o planejamento da equipe foram apresentados na Seção 3.5 como: **Velocidade**, **Estimativas Originais**, **Estimativas Finais**, **Total de Histórias Entregues** e **Total de Pontos Entregues**. Outras medidas interessantes e relacionadas ao planejamento em XP são:

- **Número de Mudanças na Iteração**: Representa o total de histórias adicionadas ou removidas pelo cliente. Por um lado, esse número pode ser baixo quando a equipe entrega software tão rápido que o cliente não tem tempo de mudar de idéia, por outro

lado, quando a iteração tem duração mais longa, esse número pode ser mais alto. Isso não é necessariamente ruim, uma vez que as equipes ágeis aceitam mudanças nos planos.

- **Erros nas Estimativas**: Quando uma equipe usa o sistema de "horas ideais" (descrito na Seção 3.5.1), é importante que aprendam a estimar de forma consistente para que a **Velocidade** se estabilize. Dessa forma, torna-se importante acompanhar o erro nas estimativas (acima ou abaixo) nas primeiras iterações de um projeto. Essa métrica deve ser descartada dentro de poucas iterações, quando a velocidade se estabilizar e o erro diminuir.

B.6. Cliente com os Desenvolvedores

Uma medida para avaliar o nível de participação e colaboração do cliente com a equipe foi apresentada na Seção B.4: a **Porcentagem dos Testes de Aceitação Passando**, uma vez que é papel do cliente definir os critérios de aceitação de uma história. Outra forma interessante de acompanhamento é medir o **Tempo Médio de Resolução de Dúvidas**: quando o cliente está presente, essa medida deve ter um valor baixo, enquanto que um cliente distante vai demorar mais tempo para responder uma dúvida.

Se a preocupação da equipe é com o cliente final do software, a Seção 3.5.2 apresentou o ***Net Promoter Score***, uma métrica que avalia a satisfação do cliente com o software produzido.

B.7. Refatoração e Design Simples

Como as práticas de Refatoração e Design Simples foram combinadas numa nova prática chamada Design Incremental, as métricas e medidas utilizadas para acompanhar essas práticas podem ser as mesmas. A Seção 3.5.1 e o estudo de caso do Capítulo 4 apresentaram diversas métricas relacionadas à qualidade do design e do código produzido, dentre elas: **WMC**, **TLOC**, **TLOTC**, $v(G)$, **LCOM**, **DIT**, **NOC**, **AC**, **CBO** e **EC**. Além dessas métricas, a

suite CK propõe outra métrica que avalia o tamanho do conjunto de métodos que podem ser chamados em resposta a uma mensagem enviada a um objeto: o ***Response for a Class*** ou RFC [30].

Outras ferramentas ajudam na avaliação do acoplamento e da qualidade de um design. Em Java, algumas ferramentas são o plug-in Eclipse Metrics[1], o JDepend[2], o Simian[3] (que tenta identificar a presença de código duplicado e também disponível para outras linguagens) e o FindBugs[4].

B.8. Padronização de Código

Para avaliar a Padronização de Código Java em relação a padrões da Sun ou outros definidos pela equipe, existem algumas ferramentas como o CheckStyle[5], o PMD[6] ou o DoctorJ[7].

B.9. Propriedade Coletiva do Código

Não é fácil encontrar uma medida objetiva e quantitativa para avaliar a Propriedade Coletiva de Código. Uma possibilidade seria analisar o repositório de código para descobrir a **Distribuição de Usuários por *Commit***. Se o projeto estiver estruturado, separando em pacotes as diferentes áreas funcionais do sistema, é possível avaliar se os usuários estão fazendo *commits* em áreas concentradas ou se essa distribuição está espalhada. Essa avaliação é mais facilmente obtida através de medidas e métricas qualitativas, como **Questionários** ou **Entrevistas**.

[1] http://metrics.sourceforge.net
[2] http://clarkware.com/software/JDepend.html
[3] http://www.redhillconsulting.com.au/products/simian
[4] http://findbugs.sourceforge.net
[5] http://checkstyle.sourceforge.net
[6] http://pmd.sourceforge.net
[7] http://doctorj.sourceforge.net

B.10. Metáfora ou Sistema de Nomes

Da mesma forma que a Propriedade Coletiva do Código, o uso da Metáfora ou Sistema de Nomes é dificilmente avaliada por métricas quantitativas. Na verdade, por se tratar de um conceito abstrato, é impossível traduzir essa prática de forma objetiva. Dessa forma, o uso de medidas e métricas qualitativas, como **Questionários** ou **Entrevistas**, é mais apropriado.

B.11. Ritmo Sustentável

Uma métrica avaliada no estudo de caso do Capítulo 4 se mostrou interessante para acompanhar se a equipe está trabalhando de forma energizada e num ritmo sustentável: o **Calendário *Niko-Niko***, ou "Humorômetro". Além disso, outras métricas já apresentadas que refletem de forma indireta a qualidade do ambiente de trabalho são a **Velocidade** (deve permanecer constante durante as iterações) e o **Número de Defeitos** (quando uma equipe faz horas extras freqüentemente, o número de falhas introduzidas no software tende a aumentar).

Outra medida mais direta para avaliar o Ritmo Sustentável é acompanhar o **Número de Horas Extras** por iteração ou outro período de tempo. Idealmente esse número deve ser baixo. Outra possível forma de minar a energia da equipe são os impedimentos que atrapalham seu progresso. Acompanhar o **Número de Impedimentos Pendentes** ou o **Tempo Médio de Resolução de Impedimentos** pode auxiliar a melhorar esse problema.

Índice Remissivo