

**Reflexões sobre o Ensino de Metodologias Ágeis  
na Academia, na Indústria e no Governo**

Alexandre Freire da Silva

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

Área de Concentração: Ciência da Computação  
Orientador: Prof. Dr. Fabio Kon

São Paulo, junho de 2007



## Agradecimentos

Agradeço meu orientador pela paciência e compreensão ao longo destes últimos 3 anos pelos quais este trabalho se alongou. Agradeço todos meus mestres, minha família e meus amigos por tudo que me ensinaram. Agradeço aos meus livros, meus discos e meu cachorro pelo apoio incondicional. I also thank the outer space structures for something that will stay between me and them. Agradeço também a Kent Beck por uma única frase que me inspirou além de todas as outras:

“A única coisa que se sabe sobre um plano é que as coisas não sairão de acordo com ele.”



## Resumo

As metodologias ágeis e em especial a Programação eXtrema surgem como um contra ponto aos métodos tradicionais de desenvolvimento de software. Nos encontramos em um momento no qual considerasse aceitável encontrar defeitos em programas de computador, até mesmo aqueles pelos quais temos que gastar muito dinheiro. Melhorar o ensino de técnicas para que equipes possam colaborar no desenvolvimento de software de qualidade é essencial para que esta área do conhecimento alcance a maturidade que esperamos.

O ensino de XP é uma tarefa relativamente complexa pois implica que pessoas passem por uma mudança cultural para aceitar seus valores, princípios e práticas. Diferentes organizações precisam adaptar a metodologia para que ela funcione bem em seu contexto local. Encontrar maneiras de facilitar o ensino e adoção das práticas ágeis é fundamental para melhorar a qualidade do software sendo desenvolvido no país.

Este trabalho pesquisa o ensino de XP em contextos acadêmicos, governamentais e industriais. Três estudos de caso foram conduzidos e analisados para sugerir padrões que podem auxiliar o ensino da metodologia por um educador em qualquer contexto.

**Palavras-chave:** Ensino, Metodologias Ágeis, Programação eXtrema, Padrões de Organização e Processo.



# Abstract

Agile methodologies and specially eXtreme Programming are born as a counter point to traditional software development methods. We find ourselves in a moment were it is considered acceptable to find bugs in computer programs, even those for which we pay a lot of money. It's is essential to make it easier to teach techniques so that teams might collaborate on the development of quality software so that this area of knowledge reaches the maturity we hope for.

Teaching XP is a relatively complex task because it implies that people must go through a cultural change to accept it's values, principles and practices. Different organizations need to adapt the methodology so that it will work well in their local context. Finding ways to facilitate teaching and adopting agile practices is fundamental to improve the quality of software being developed in the country.

This work researches the process of teaching XP in academic, governmental and industrial contexts. Three case studies were conducted and analyzed so that we could suggest patterns that can support educators teaching the methodology in any context.

**Keywords:** Teaching, Agile Methodologies, eXtreme Programming, Organizational and Process Patterns.



# Sumário

|  |           |
|--|-----------|
| <b>Lista de Figuras</b>                        | <b>xi</b> |
| <b>1 Introdução</b>                            | <b>1</b>  |
| 1.1 Objetivos deste trabalho . . . . .         | 6         |
| 1.2 Não são objetivos deste trabalho . . . . . | 8         |
| 1.3 Trabalhos relacionados . . . . .           | 8         |
| <b>2 Uma reflexão sobre XP</b>                 | <b>11</b> |
| 2.1 XP - Como funciona . . . . .               | 13        |
| 2.2 XP - Por que funciona? . . . . .           | 16        |
| 2.3 Valores . . . . .                          | 19        |
| 2.3.1 Comunicação . . . . .                    | 19        |
| 2.3.2 Simplicidade . . . . .                   | 21        |
| 2.3.3 <i>Feedback</i> . . . . .                | 21        |
| 2.3.4 Coragem . . . . .                        | 22        |
| 2.4 Práticas . . . . .                         | 23        |
| 2.4.1 Jogo do planejamento . . . . .           | 24        |
| 2.4.2 <i>Releases</i> pequenos . . . . .       | 26        |
| 2.4.3 Metáfora . . . . .                       | 27        |

|          |  |           |
|----------|--|-----------|
| 2.4.4    | <i>Design</i> simples . . . . .  | 28        |
| 2.4.5    | Testes automatizados . . . . .   | 28        |
| 2.4.6    | Refatoração . . . . .  | 29        |
| 2.4.7    | Programação pareada . . . . .  | 30        |
| 2.4.8    | Propriedade coletiva do código . . . . .   | 31        |
| 2.4.9    | Integração contínua . . . . .  | 32        |
| 2.4.10   | Ritmo sustentável . . . . .  | 33        |
| 2.4.11   | Cliente sempre presente . . . . .  | 34        |
| 2.4.12   | Padrão de codificação . . . . .  | 35        |
| 2.5      | Papéis . . . . .   | 36        |
| 2.5.1    | Treinador . . . . .  | 36        |
| 2.5.2    | Acompanhador . . . . .   | 37        |
| 2.6      | Refatorações e a adoção de novas práticas . . . . .  | 39        |
| 2.7      | A nova versão de XP . . . . .  | 41        |
| 2.7.1    | Princípios . . . . .   | 42        |
| 2.7.2    | Práticas primárias . . . . .   | 45        |
| 2.7.3    | Práticas corolário . . . . .   | 47        |
| 2.7.4    | Papéis . . . . .   | 49        |
| 2.7.5    | Mudanças e evoluções . . . . .   | 50        |
| <b>3</b> | <b>Experiências com o ensino de XP</b> . . . . .   | <b>53</b> |
| 3.1      | Trabalhos relacionados na Academia . . . . .   | 54        |
| 3.2      | Laboratório de XP - 2001 a 2007 . . . . .  | 58        |
| 3.2.1    | Como funciona o Laboratório . . . . .  | 59        |
| 3.2.2    | Mico - Sistema para administração de carga didática - nossa primeira experiência . . . . . | 66        |

|          |  |           |
|----------|--|-----------|
| 3.2.3    | Marcador de Reuniões - grupo pequeno utiliza Smalltalk . . . . .                                   | 69        |
| 3.2.4    | Colméia - Gerenciador de Biblioteca - evolução de um projeto ao longo dos anos                     | 70        |
| 3.2.5    | Cigarra - Distribuidor Multimídia - clientes externos em um grande projeto governamental . . . . . | 71        |
| 3.3      | Trabalhos relacionados na Indústria . . . . .  | 74        |
| 3.4      | Paggo - uma <i>start-up</i> brasileira adota XP . . . . .  | 79        |
| 3.5      | Trabalhos relacionados no Governo . . . . .  | 84        |
| 3.6      | Ministério da Cultura - O projeto Cultura Digital . . . . .  | 87        |
| 3.7      | <i>Workshops</i> , cursos introdutórios, jogos e eventos com a comunidade . . . . .                | 94        |
| <b>4</b> | <b>Análise e reflexão sobre o ensino de XP</b>   | <b>99</b> |
| 4.1      | O cenário ideal . . . . .  | 100       |
| 4.2      | Etapa inicial - Começando o processo de transição . . . . .  | 102       |
| 4.2.1    | Padrões para convencer sua organização a praticar XP . . . . .                                     | 103       |
| 4.2.2    | Padrões para escolher um treinador e se envolver realmente com o cliente . . .                     | 107       |
| 4.2.3    | Padrões para montar uma equipe . . . . .   | 109       |
| 4.2.4    | Padrões para estruturar o espaço de trabalho . . . . .   | 111       |
| 4.2.5    | Padrões para o planejamento inicial . . . . .  | 113       |
| 4.3      | Aprendizado através da repetição prática - Amadurecimento da metodologia . . . . .                 | 116       |
| 4.3.1    | Padrões de treinamento . . . . .   | 117       |
| 4.3.2    | Padrões de planejamento contínuo . . . . .   | 120       |
| 4.3.3    | Padrões de <i>design</i> . . . . .   | 122       |
| 4.3.4    | Padrões para aplicar no dia-a-dia . . . . .  | 125       |
| 4.4      | Etapa final - A equipe desenvolve sua própria metodologia . . . . .                                | 128       |
| 4.4.1    | Observações Póstumas . . . . .   | 129       |

|  |            |
|--|------------|
| 4.5 Práticas Ágeis em contextos não ligados ao desenvolvimento de software . . . . . | 131        |
| <b>5 Conclusões</b>  | <b>135</b> |
| 5.1 Principais contribuições . . . . .   | 136        |
| 5.2 Trabalhos futuros . . . . .  | 136        |
| <b>Referências Bibliográficas</b>  | <b>137</b> |

## Lista de Figuras

|      |  |    |
|------|--|----|
| 2.1  | As práticas se apóiam (BECK, 1999) . . . . .   | 23 |
| 2.2  | Histórias implementadas (em verde), mudanças em histórias (em amarelo) e correção de defeitos (em vermelho) realizadas em cada iteração (cada coluna é uma iteração). (JEFFRIES, 2004) . . . . . | 38 |
| 2.3  | A evolução das práticas de XP . . . . .  | 51 |
| 3.1  | O espaço do laboratório. . . . .   | 60 |
| 3.2  | Uma equipe ocupa suas estações de programação pareada no laboratório. . . . .  | 61 |
| 3.3  | Outro par desenvolvendo seu sistema. . . . .   | 62 |
| 3.4  | O quadro branco de uma equipe sendo utilizado como radiador de informações. . . . .  | 63 |
| 3.5  | Radiadores de informações colados na parede. . . . .   | 64 |
| 3.6  | Dois professores que também atuam como clientes refletem sobre os projetos em andamento durante o almoço. . . . .  | 65 |
| 3.7  | Histórias de uma equipe on-line no XPlanner. . . . .   | 66 |
| 3.8  | Enfermeiras clientes do Borboleta acompanham a apresentação de um release. . . . .   | 67 |
| 3.9  | Evolução de número de testes de uma equipe detalhada numa tabela pelo seu acompanhador. . . . .  | 68 |
| 3.10 | O treinador da equipe Cigarra atuando. . . . .   | 71 |
| 3.11 | Reunião de apresentação do primeiro <i>release</i> da equipe Cigarra. . . . .  | 73 |

|      |  |     |
|------|--|-----|
| 3.12 | Dois pares trabalhando na Paggo. . . . .   | 80  |
| 3.13 | Acompanhadora atualizando radiadores de informação Paggo. . . . .  | 81  |
| 3.14 | Radiador de informação com acompanhamento de um dos primeiros <i>releases</i> de sistemas da Paggo. . . . .  | 83  |
| 3.15 | Quadro de histórias para acompanhamento de um <i>release</i> na Paggo. . . . .   | 84  |
| 3.16 | Radiador de informação mostrando evolução da base de código de um dos sistemas da Paggo. . . . .   | 85  |
| 3.17 | Cliente <i>proxy</i> pareando com um desenvolvedor na Cultura Digital. . . . .   | 89  |
| 3.18 | Cliente <i>proxy</i> durante jogo do planejamento na Cultura Digital. . . . .  | 91  |
| 3.19 | Quadro de Histórias de iterações da Cultura Digital. . . . .   | 92  |
| 3.20 | Programação pareada no novo espaço da Cultura Digital. . . . .   | 93  |
| 3.21 | Jogadores em um XP Game trabalham para resolver uma história. . . . .  | 96  |
| 4.1  | O padrão <b>Mapa Mental</b> pode ser usado para entender melhor a metodologia. Neste mapa mental vemos práticas relacionadas ao planejamento de um projeto. . . . .                    | 104 |
| 4.2  | O anti-padrão <b>Personalidade Múltipla</b> pode ser resolvido com um chapéu. . . . .  | 109 |
| 4.3  | Um mural mostra o planejamento ágil de uma oficina de conhecimentos livres. Participantes podiam propor mudanças no cronograma e até mesmo novas atividades durante a oficina. . . . . | 133 |
| 4.4  | Retrospectivas são úteis para qualquer equipe que trabalha de maneira colaborativa. . . . .  | 134 |

# Capítulo 1

## Introdução

Atualmente, não há dúvidas que a indústria de software se estabeleceu como uma das mais importantes. Computadores, cada vez menores e mais rápidos, se espalham ubiquamente pelo mundo, presentes em grande parte das ferramentas usadas pelos seres humanos, de geladeiras a telefones celulares, de carros a casas. Empresas, sejam elas micro, pequenas, médias, grandes ou gigantes do mercado, investem em computadores e programas para conseguir sobreviver na era da tecnologia da informação. O mercado do desenvolvimento é um dos únicos no qual a demanda por trabalhadores continua maior do que a oferta de programadores. Num mundo onde o segundo homem mais rico do planeta alcançou esta posição vendendo software, não há mais como questionar a importância desta prática produtiva. Constatamos que melhorar o ensino e divulgação de métodos de desenvolvimento de software é importante tanto para a própria indústria, quanto para a sociedade como um todo. Para tornar o Brasil um país de destaque no mercado global de software, precisamos refletir sobre a produção de software, como se dá esse processo hoje e como podemos melhorá-lo.

É natural pensar que uma tecnologia tão recente quanto o computador ainda não tenha encontrado maturidade suficiente para podermos dizer que sabemos tudo sobre a natureza da produção de software. A disciplina de Engenharia de Software surgiu há apenas 38 anos, desde então diversos estudos são realizados para tentar tornar a criação de software mais eficiente e menos propensa a erros. Mesmo assim, não temos experiência suficiente nesta arte se comparada ao conhecimento adquirido em outras disciplinas mais maduras, a engenharia civil, por exemplo, tem suas raízes nas primeiras civilizações humanas. Aprendendo com a experiência dos romanos, hoje em dia construímos pontes muito mais baratas e eficientes.

Sabemos que, por mais esforço e dinheiro que se aplique nesta indústria, grande parte dos projetos

de software hoje em dia fracassa. O software produzido é defeituoso, inadequado aos desejos do cliente e é entregue fora do prazo e acima dos custos esperados. O último *CHAOS Report* [140], estudo envolvendo milhares de projetos na área de Tecnologia de Informação, revela que 51% deles se encontra em situação de risco, 15% fracassaram e somente um terço, 34%, foi bem sucedido. No total, 43% dos projetos custaram acima do esperado e 82% foram entregues fora do prazo. Somente 52% das funcionalidades desejadas chegam a ser implementadas no produto entregue.

Uma das principais causas de tantos fracassos é obviamente a falta de qualificação dos profissionais, como apontam alguns estudos [1,120,152,35]. Grande parte da força de trabalho, especialmente em países em desenvolvimento como o Brasil, é composta por pessoas com pouca instrução formal, são auto-didatas, que através de documentação e exemplos disponíveis em livros e na Internet conseguiram dominar o necessário da técnica para serem aceitos no mercado de trabalho. Mercado este que, infelizmente, adota linguagens de programação e métodos movidos mais pela força do marketing das empresas que as desenvolveram (e conseqüente disponibilidade de desenvolvedores que as conhecem), do que devido aos méritos técnicos, além de não incentivar e apoiar o crescimento de seus funcionários.

Programar, o ato de codificar instruções em alguma linguagem, que por sua vez serão processadas em um computador, é uma atividade que vem sendo exercida por um número cada vez maior de pessoas. Com a disseminação dos computadores, programar deixou de ser uma atividade exclusiva de engenheiros ou cientistas da computação. Existem muitas ferramentas e técnicas que pretendem facilitar a vida dos programadores. Os programas livres e de código aberto oferecem a todos excelentes ferramentas, arcabouços e fontes de inspiração e aprendizado. Comunidades internacionais se organizam para trocar informações e ajudar novos desenvolvedores. Empresas investem em novos processos e metodologias, cujas opções variam de acordo com o tamanho do projeto, tamanho da equipe de desenvolvedores e até mesmo a exigência de algum tipo de certificação.

A falta de educação de qualidade acessível aos programadores é uma das principais causas de falhas, aqui iremos abordar o ensino de metodologias ágeis, suas práticas e processos, desta maneira podemos melhorar a qualidade do trabalho dos desenvolvedores. Acreditamos ser esta uma contribuição valiosa deste trabalho para a comunidade.

Poucos dos métodos de desenvolvimento concentram seus esforços nas pessoas, nos programadores. Usando a metáfora comum de *Engenharia* de Software, muitos tratam o software como qualquer outro bem industrial e o programador como um operário, mais uma peça da linha de produção.

Para produzir software, inspiram-se em técnicas nascidas na revolução industrial. Alvin Toffler [149] escreve sobre essa revolução:

“Um trabalhador único tradicional, efetuando todas as operações necessárias sozinho, podia produzir apenas um punhado de alfinetes por dia, não mais de 20 e talvez nem um. Em contraste, numa manufatura, na qual se exigiam 18 operações diferentes efetuadas por dez trabalhadores especializados, cada um efetuando apenas uma ou algumas fases, juntos conseguiam produzir mais de 48 mil alfinetes por dia, mais de quatro mil e oitocentos por trabalhador.”

(TOFFLER, 2001, p.62).

Estes métodos acreditam que dividir equipes e responsabilizar pequenos grupos por tarefas especializadas é o caminho para criar uma fábrica de software eficiente. Como o software não é um bem material, ao encaminhá-lo de uma posição a outra na linha de montagem, estes métodos ditam que vários artefatos e documentação detalhada devem acompanhar o código. Um longo período de planejamento é necessário para especificar cuidadosamente o objetivo do software e depois criar uma arquitetura que será dividida em pedaços que poderão ser então codificados, integrados e, finalmente, testados e enviados ao cliente.

Estes métodos pecam por acreditar que o trabalho criativo, o desenvolvimento de software, possa ser otimizado neste modelo desenvolvido para acelerar a produção de bens materiais. “Engenharia” é uma metáfora que foi empregada ao desenvolvimento de software de maneira equivocada. Diversos autores discutem a diferença entre os trabalhadores manuais e os trabalhadores criativos (ou trabalhadores do conhecimento). Domenico de Masi [41] exemplifica:

“Se sou um publicitário e estou tentando criar um *slogan*, quando saio do escritório e volto para casa, levo o trabalho comigo: na minha cabeça. A minha cabeça não pára de pensar e às vezes acontece que posso achar a solução para o slogan em plena noite, ou debaixo do chuveiro, ou ainda naquele estado intermediário entre o sono e o despertar”

(MASI, 2000, p.205)

Kent Beck, na segunda edição de seu livro sobre Programação eXtrema [19], cita como base filosófica da metodologia algumas falhas de aceitar a metáfora da engenharia, herança da revolução industrial, ao desenvolvimento de software:

“Enquanto o Taylorismo tem alguns efeitos positivos, também tem sérias deficiências. Essas limitações tem origem em três preconceitos:

1. As coisas normalmente acontecem como planejado.
2. Micro-otimização leva à macro-otimização.
3. Pessoas podem ser substituídas e precisam receber ordens para trabalhar.”

(BECK, 2004, p.132)

O autor discute a importância do Taylorismo, movimento liderado por Frederick Taylor [145] que impulsionou a revolução industrial, relativa ao desenvolvimento de software. Beck aponta a existência de duas mudanças sociais implícitas nesta revolução. A primeira é a separação entre planejamento e execução. Trabalhadores devem executar a tarefa delegada fielmente, da maneira pré-estipulada e no tempo determinado previamente. A segunda é a criação de um departamento de qualidade separado, o que implica que qualidade não é responsabilidade de todos. Beck conclui:

“... estruturas sociais Tayloristas impedem o fluxo de comunicação e *feedback* vitais para criação de software funcional, flexível e barato, em um mundo que está em constante mudança.”

(BECK, 2004, p.133)

Martin Fowler [51] também critica a separação social entre engenheiros e arquitetos e programadores:

“Na construção civil há mais clareza na separação de habilidades entre aqueles que planejam e desenham e os que constroem, mas esse não é o caso no desenvolvimento de software.

Qualquer programador trabalhando em um ambiente de *design* complexo precisa ser habilidoso o suficiente para questionar o *design* do arquiteto, especialmente quando este tem menos conhecimento sobre as realidades do dia-a-dia do desenvolvimento da plataforma.”

(FOWLER, 2000, p.1)

Até mesmo no chão de fábrica, provou-se que questionar essas premissas leva ao aumento da produtividade, o Sistema de Produção da Toyota [107], inovou e quebrou recordes produtivos ao eliminar o departamento de qualidade, tornando todos trabalhadores responsáveis por ela.

Mesmo assim, muitas empresas no ramo de desenvolvimento de software insistem em usar técnicas oriundas da revolução industrial ou de áreas como a construção civil. Uma delas, bastante comum, é a de associar bônus remunerativos ao aumento da produção, para incentivar seus empregados. Existem estudos econômicos [48, 49, 58, 59] que comprovam que este tipo de contrato de incentivo, no âmbito da produção criativa, não funciona, reduzindo a produtividade e a cooperação voluntária. Yochai Benkler [21] aponta que a construção de boas relações sociais e incentivos ao compartilhamento de conhecimento tem efeito positivo no aumento da produtividade e na cooperação.

Foi pensando nisso, na construção de um ambiente de trabalho melhor e mais produtivo, que surgiram as metodologias ágeis de desenvolvimento de software. No começo de 2001, diversos agentes subjacentes a métodos novos se reuniram e escreveram o *Manifesto Ágil*. Entre os métodos representadas estavam: *Adaptive Software Development*, *Programação eXtrema (XP)*, *Scrum*, *Crystal*, *Feature Driven Development (FDD)*, *Dynamic System Development Method (DSDM)*, *Lean Software Development* e *Pragmatic Programming*. No manifesto, todos concordaram em alguns valores comuns: as metodologias ágeis concentram-se nas pessoas envolvidas na produção, assumem que planejamentos a longo prazo são sempre falhos e que o importante é ser ágil para poder lidar com mudanças entregando, continuamente, software funcional. Em comparação com métodos tradicionais o conjunto de técnicas e práticas dos métodos ágeis é menor e mais simples, fazendo com que a sua adoção seja relativamente fácil para organizações interessadas. Além disso, o acompanhamento das atividades deixa de ser subjetivo, tornando mais fácil prestar conta do progresso em um projeto.

O manifesto ágil [4] diz:

“Estamos trazendo à tona novas e melhores maneiras de desenvolver software desenvolvendo-o e ajudando outros a desenvolvê-lo. Através deste trabalho viemos a valorizar:

- **Indivíduos e interações** ao invés de processos e ferramentas

- **Software funcional** sobre documentação completa e detalhada
- **Colaboração com o cliente** sobre negociações de contrato
- **Adaptação a mudanças** sobre seguir planos

Isto é, enquanto os itens na direita têm valor, nós damos mais valor aos itens na esquerda.”

(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, Dave Thomas, 2001)

Uma das primeiras metodologias ágeis propostas foi a *Programação eXtrema (XP)*. XP inova com um conjunto pequeno de práticas que podem ser adotadas rápida e efetivamente por organizações de pequeno a médio porte, aumentando a produtividade da organização e a qualidade do software produzido.

XP se concentra nas atividades dos programadores e a maioria das práticas são voltadas para eles. A Programação eXtrema também dá muita atenção ao escopo do software a ser produzido, prometendo entregar exatamente o que o cliente precisa no menor tempo e com o menor custo possível.

Acreditamos que nos poucos anos que o ato de produzir software teve para amadurecer, as metodologias ágeis inovam indicando um caminho mais natural para o trabalhador criativo, proporcionando melhores condições para que a indústria possa produzir software de qualidade e para que o criador possa exercer seu trabalho de maneira mais humana.

## 1.1 Objetivos deste trabalho

Esta dissertação tem como objetivo analisar experiências de ensino e implantação de XP tanto na universidade quanto em empresas privadas e projetos governamentais. Iremos refletir sobre as práticas propostas pela metodologia e maneiras de ensinar grupos de estudantes e programadores a adotá-las no seu dia-a-dia. Mudar a cultura de um grupo nem sempre é tarefa simples, ainda mais quando propomos práticas como as de XP, que geram resistência em boa parte das pessoas. Acreditamos que o ensino destas metodologias não é algo trivial, iremos mostrar como valores e práticas de outros métodos ágeis e a própria evolução da Programação eXtrema ao longo dos anos, nos ajudam a observar padrões que podem facilitar esta tarefa.

Iremos começar com uma descrição reflexiva da metodologia, apresentando em mais detalhes a sua primeira versão, discutindo e analisando brevemente cada um dos valores, das práticas e dos papéis exercidos pelos membros de uma equipe XP. Nesta discussão pretendemos refletir sobre os detalhes peculiares de cada prática e seus efeitos, assim como a relação entre diferentes práticas, valores e papéis. Iremos também considerar a evolução da metodologia para sua segunda versão oficial, um refinamento dos valores, práticas, princípios e papéis realizado por Beck após cinco anos de amadurecimento e pesquisa metodológica.

Em seguida, pretendemos fazer um relato dos casos que estudamos neste trabalho. Apresentaremos uma síntese das diversas experiências observadas, começando com um apanhado de evidências na literatura, em seguida descrevendo com maiores detalhes um caso de estudo nosso em cada um dos diferentes contextos abordados. Em primeiro lugar iremos apresentar cinco anos de experiência no ensino de grupos de alunos universitários, de graduação e pós-graduação, cada qual trabalhando em um projeto específico dentro do escopo de uma disciplina optativa com duração de um semestre, o Laboratório de Programação eXtrema do IME/USP [63]. Em segundo lugar, discorreremos sobre a experiência de implantar XP em uma organização privada, na qual todos os funcionários da empresa Paggo aprenderam as técnicas e valores de XP e passaram a adotar Programação eXtrema como sua metodologia. Em terceiro lugar, iremos refletir sobre a experiência de implantar XP em alguns projetos de cunho governamental no Ministério da Cultura. Finalmente, iremos falar sobre a experiência de participar de eventos da comunidade, organizar palestras e jogos, ministrar cursos de curta duração e oferecer pequenas consultorias com o intuito de divulgar métodos ágeis.

A seguir iremos apresentar uma análise e reflexão sobre a experiência do ensino desta metodologia nesses contextos diferenciados. Pretendemos resumir questões relacionadas a cada uma das práticas em cada caso, além de apresentar novas práticas que utilizamos, as razões por fazê-lo e os efeitos observados. Vamos apresentar intervenções nos diferentes papéis que fazem parte da metodologia e discutir o mérito e a eficácia de tais intervenções nos diferentes casos. Iremos ainda discutir as maiores dificuldades e problemas encontrados, apresentando sugestões a partir de nossa análise qualitativa que evidência as diferenças entre os contextos. Durante esta exposição, iremos refletir sobre o ensino desta metodologia, mencionando práticas e idéias como padrões que podem ser adotados por outros, além de anti-padrões que devem ser evitados, possivelmente ajudando futuros processos de implantação de XP em ambientes similares aos estudados. Iremos também discutir a adoção de práticas da metodologia em outras áreas de uma organização, não necessariamente técnicas, refletindo sobre duas experiências do tipo.

Finalmente, iremos apresentar as conclusões às quais chegamos através deste trabalho.

## 1.2 Não são objetivos deste trabalho

Não é objetivo deste trabalho introduzir XP em detalhes ao leigo ou apresentar os vários métodos ágeis, para isso recomendamos a leitura dos livros introdutórios sobre os temas. [16] e [19] introduzem XP. Os outros métodos ágeis: *Adaptive Software Development*, *Scrum*, *Crystal*, *Feature Driven Development (FDD)*, *Dynamic System Development Method (DSDM)*, *Lean Software Development* e *Pragmatic Programming* são introduzidas em [74, 136, 30, 118, 141, 121] e [73].

Além disso, não iremos descrever experimentos científicos controlados ou pesquisas para avaliar a eficácia de XP e outros métodos ágeis. Assim como não pretendemos ter feito um trabalho abrangente e não fizemos amostras suficientes para generalizar soluções em qualquer contexto acadêmico, industrial ou governamental. Outros estudos foram realizados de forma a coletar dados de experiências com XP, para avaliar o sucesso de projetos e a eficácia de práticas, analisando dados qualitativos e quantitativos para validar os métodos ágeis em diferentes contextos. Interessados nessas análises podem ler [153, 148, 127, 122, 143, 106, 138, 129, 87, 83, 88, 158, 131] e [130].

No mais, não pretendemos descrever software desenvolvido com XP. Interessados podem ler nossos estudos que concentram sua análise nos programas criados [54, 57] e [53].

O fator psicológico de satisfação no trabalho é um aspecto interessante que inicialmente pensamos em abordar, interessados podem ler estudos que levam em conta o uso de métodos ágeis do ponto de vista psicológico e de motivação no trabalho [27, 94, 12] e [102].

## 1.3 Trabalhos relacionados

Apesar de toda atenção que os métodos ágeis vêm recebendo atualmente, eles são relativamente recentes e não chegaram a completar nem uma década de existência.

A série de livros introdutórios de XP, apelidada de *XP Series*, oito livros que falam em detalhes sobre a metodologia [16, 19, 89, 43, 79, 155, 126] e [80], além do livro sobre metodologias ágeis de Cockburn [29], são a base teórica de onde obtivemos grande parte do conhecimento necessário para ensinar as práticas e técnicas das metodologias ágeis. Destes livros, apenas o clássico de Kent Beck [16] foi traduzido, existe apenas um livro sobre a metodologia XP [146] de autores brasileiros.

As referências estão presentes ao longo da dissertação, mas julgamos interessante apresentá-las

rapidamente, separadas em tópicos nesta seção.

Interessados em casos de sucesso de uso de XP podem ler uma tese sobre uma experiência brasileira [147], muitos artigos também relatam casos de sucesso de adoção da metodologia [119, 90, 135, 42, 60, 24, 71, 86, 11, 75] e [55].

Muitos trabalhos foram efetuados sobre as práticas de XP isoladamente. Interessados podem ler sobre programação pareada, prática anterior à XP, muito aprofundada no trabalho de Laurie Williams [160, 161, 162, 28, 157] e [159]. Outros autores também abordam a programação pareada: [150, 93, 114, 72] e [91]. Por ser de muita importância na metodologia, o papel do cliente também já foi bastante estudado: [64, 47, 156, 95, 82, 96, 98] e [97]. Testes automatizados tem sua própria literatura também, a começar pelo trabalho sobre desenvolvimento dirigido por testes de Kent Beck [15, 18], passando por avaliações e casos de outros autores: [108, 112, 92, 62], até literatura específica sobre testes de aceitação [33, 9] e testes de interfaces gráficas [103, 104]. Histórias já foram estudadas [10, 105] e existem trabalhos que aprofundam-se sobre aprendizado de estimativas [23]. Em [125] discute-se como escalar a prática de integração contínua. Refatoração também é anterior à XP e tem pesquisa própria: [50, 6] e [99].

Ensinar XP, o tópico do nosso trabalho, também é um tópico bem discutido na academia, alguns artigos interessantes abordam esse desafio de diferentes pontos de vista: [13, 163, 70, 111, 85, 151, 139, 68, 20, 133, 144, 117, 109, 67, 65, 44, 101, 113, 110, 66, 63, 116, 45] e [69].

São raros trabalhos relatando casos de fracasso de XP, pela eficácia da metodologia e pelos fracassos serem relacionados a um processo falho de adoção do método, ou a adoção parcial do método [14]. Como relata [77]: o método requer investimento no aprendizado prático, provocando críticas de pessoas que não estão dispostas a mudar os seus valores, comportamentos e cultura.

Trabalhos sobre falhas, fracassos e críticas existem: [22, 32, 78, 134]. Um pouco antes de Beck publicar a sua reformulação da metodologia, dois livros [100, 142] foram publicados com críticas e propostas de mudanças em aspectos da metodologia.



## Capítulo 2

### Uma reflexão sobre XP

Programação eXtrema surgiu na indústria, quando em 1997, Kent Beck e um grupo de oito consultores das comunidades de padrões e orientação a objetos foram chamados para salvar um projeto de software da Chrysler [16]. Era o sistema de folha de pagamento, estava atrasado e já tinha custado muito mais do que deveria. Durante um ano, uma equipe de vinte e seis pessoas não conseguiu entregar uma versão funcional do software. No ano seguinte, Kent Beck e sua equipe conseguiram. Eles experimentaram uma metodologia diferente, linguagens de padrões organizacionais e de processo que foram reconhecidos pela comunidade, assim nasceu o primeiro caso de sucesso de XP. Em 1999, Beck escreveu seu primeiro livro sobre a metodologia, optando por usar a palavra *extreme* para chamar atenção dos programadores ao fato de que o novo método tentava aplicar todas as práticas reconhecidamente boas da programação e levá-las ao extremo. Por exemplo, se revisão de código é uma boa prática, em Programação eXtrema todo o código escrito será revisado enquanto está sendo criado, é a prática extrema de programação pareada. Outro exemplo, se testar o código é uma boa prática, iremos testar todo o código antes mesmo dele ser escrito, é a prática extrema de testes automatizados. A marca se tornou popular rapidamente e hoje em dia é uma metodologia reconhecida tanto na indústria, quanto na academia. A segunda edição do livro de Beck [19] é uma evolução completa (até as práticas foram repensadas) e foi escrita com uma linguagem mais positiva e inclusiva em 2004, levando em consideração cinco anos de experiência da comunidade ágil e as críticas recebidas por causa da linguagem mais incisiva do primeiro livro.

O leitor deve ter percebido que até agora utilizamos tanto a palavra “método” quanto “metodologia” para nos referirmos à Programação eXtrema, cabe um esclarecimento sobre a terminologia adotada. Ao pesquisar a palavra “metodologia” em dicionários, vemos que ela é definida de maneiras

distintas. Uma maneira é “a arte de dirigir o espírito na investigação da verdade”, bela definição, porém longe do uso comum [38]. Em outra definição vemos um ‘corpo de regras e diligências estabelecidas para realizar uma pesquisa’, ou seja, como no uso coloquial comum, metodologia é sinônimo de “método”, mas pode ser ainda “parte de uma ciência que estuda os métodos aos quais ela própria recorre” [37]. Neste trabalho pretendemos elaborar uma reflexão sobre o ensino de métodos ágeis, ao fazê-lo iremos eventualmente classificar estes métodos, e XP com mais frequência, como metodologias, para ressaltar o estudo do próprio método como prática importante desta ciência. Ou seja, XP, como uma metodologia, define procedimentos de ensino da arte de programar software de qualidade coletivamente e, ao mesmo tempo, propõe o estudo científico dos próprios métodos adotados, podendo adaptá-los e evoluí-los de acordo com o contexto local no qual será aplicada.

XP nasceu como fruto das experiências e descobertas das comunidades de Smalltalk e padrões de design orientado a objetos. Após estudar os próprios métodos, desenvolvedores que utilizavam orientação a objetos generalizaram padrões de design [61], onde um padrão é uma solução reconhecida para um problema comum. Uma linguagem de padrões é uma receita de maneiras possíveis de usar padrões em combinação [2]. XP pode ser entendida como uma metodologia que propõe e estuda um conjunto de linguagens de padrões de organização e processo [7, 8, 26, 31, 124, 25, 84, 56]. Nada mais natural então do que esta metodologia de desenvolvimento tenha sido criada nesta comunidade.

Cockburn [29] define uma metodologia como “as convenções com as quais um grupo concorda” e detalha os elementos que a compõem:

- Equipes de pessoas que entram em acordo sobre um conjunto de valores e princípios.
- Pessoas ocupando diferentes papéis na equipe, estes demandam habilidades e personalidades diferentes.
- Atividades realizadas no dia-a-dia da equipe, empregando diferentes técnicas, podendo gerar artefatos ou não.
- Um processo que diz como essas diferentes atividades interagem no tempo, quais são os eventos que marcam progresso e quais são as convenções seguidas pela equipe.
- A avaliação da qualidade dos produtos gerados e das atividades realizadas pela equipe.

Os padrões organizacionais e de processo são aplicados neste contexto e se dividem em linguagens que cobrem os diferentes aspectos da produção de software. Os padrões de processo descrevem

abordagens de sucesso comprovadas pela comunidade, além de uma série de atividades que podem ser executadas ao desenvolver software. Os padrões organizacionais descrevem como criar e adaptar processos junto às organizações, como funcionam estes processos, como os diferentes papéis da equipe se relacionam, quais atividades são realizadas no trabalho e quais são os artefatos produzidos. Um estudo recente [36] mostra que vários destes padrões são reconhecidos nas metodologias ágeis.

A seguir, iremos conhecer os padrões que fazem parte de XP e observar como a linguagem de padrões subjacente à metodologia evoluiu ao mesmo tempo que o estudo de seus próprios métodos continuou durante cinco anos. Primeiro iremos apresentar o funcionamento da metodologia de acordo com a sua primeira versão e depois explicitar as razões que justificam a eficácia desta de acordo com a segunda versão do livro de Beck. Então, iremos entrar em detalhes sobre os valores, práticas e papéis da primeira versão. Ao apresentar cada prática iremos discorrer sobre o relacionamento entre as práticas. Iremos também mencionar a adoção de outras práticas ágeis e a criação de práticas personalizadas. Finalmente iremos comentar as mudanças na segunda versão de XP, seus novos valores, princípios, práticas e papéis.

## 2.1 XP - Como funciona

XP é indicado para equipes pequenas e médias, que desenvolvem software baseado em requisitos não totalmente definidos e que se modificam rapidamente, por clientes com projetos não-críticos de complexidade baixa ou média. Na primeira versão de seu livro Beck define:

“Programação eXtrema (XP) é uma metodologia leve para times pequenos e médios desenvolvendo software com requisitos vagos ou que mudam rapidamente.”

(BECK, 1999, p.1)

A metodologia é leve, pois os processos definidos minimizam esforços burocráticos e pequena, contendo apenas doze práticas e quatro papéis. Por isso, pode ser posta em prática em um período curto de tempo, sem grandes custos às empresas. É uma metodologia heurística e tolerante a adaptações, facilitando com que a instituição aprenda com o passado e adapte a metodologia ao seu contexto. *Algumas práticas requerem muita disciplina. Muitas práticas são interdependentes, completando-se e apoiando-se, por isto, podem existir riscos na adoção de somente uma parte do conjunto de práticas.* Por ter sido uma metodologia criada por programadores, a maioria das práticas focaliza sua atenção

neles e no ato de programar.

Em XP, a equipe valoriza a **comunicação** entre as pessoas, a **simplicidade** do software e do próprio processo, o **feedback** constante e contínuo e a **coragem**.

Os valores se concretizam em alguns princípios básicos, evidenciados nas práticas da metodologia. O primeiro princípio é **velocidade no feedback**, o tempo entre uma ação e o seu devido *feedback* deve ser o mais curto possível, para assim melhorar o processo de aprendizado. **Assumir simplicidade** implica que ao se deparar com um problema, as pessoas devem assumir que ele é fácil de ser resolvido e procurar uma solução simples. **Mudança incremental** tenta evitar grandes mudanças repentinas, pois elas simplesmente não funcionam, priorizando pequenas mudanças, realizadas aos poucos. **Acolher mudanças** é o princípio de seguir a estratégia de aceitar mudanças que resolvem o problema mais importante que existe em um momento, ao mesmo tempo preservando suas opções. O último princípio básico é **trabalho de qualidade**, pois uma das necessidades secundárias dos seres humanos é obter satisfação por realizar trabalhos bem feitos. Além desses princípios básicos, alguns princípios menos importantes são: ensinar a aprender, começar com investimento pequeno, jogar para ganhar, experimentos concretos, comunicação honesta e aberta, trabalhar com os instintos da pessoas, responsabilidade aceita, adaptação local, viajar sem peso e medidas honestas.

Na equipe observamos quatro papéis principais: os programadores, o cliente, um treinador e um acompanhador, além de eventuais consultores.

O papel de treinador é importante para uma equipe conseguir mudar sua cultura de desenvolvimento e se adaptar à metodologia corretamente. As práticas de XP demandam muita disciplina e o treinador é a pessoa responsável por ensiná-las à equipe e garantir que sejam executadas corretamente.

O acompanhador mantém dados e métricas relativos ao andamento do processo e a qualidade dos artefatos gerados, seu trabalho é essencial como suporte ao treinador. Comunicando as métricas à equipe, consegue manter todos conscientes de como o trabalho está andando de fato e onde e como pode-se melhorar.

As práticas da primeira versão são:

- jogo do planejamento

- *releases* curtos
- metáfora
- *design* simples
- testes automatizados
- refatoração
- programação pareada
- propriedade coletiva do código
- integração contínua
- ritmo sustentável
- cliente sempre presente
- padrão de codificação

A segunda versão contempla todas estas práticas, mas muda alguns nomes e introduz novas práticas, iremos comentar estas mudanças mais tarde.

O desenvolvimento é feito em ciclos de algumas semanas, chamados de iterações. Cada iteração produz software funcional, integrado e testado, contendo as funcionalidades mais necessárias aos clientes. **Releases curtos** contém algumas iterações, mas por serem curtos, duram no máximo um mês. Ao entregar um *release*, o software deve ser colocado em produção. A sobrecarga de trabalho é evitada, ao selecionar para a iteração uma carga de trabalho com a qual a equipe se compromete de fato. Além disso, as iterações tentam adotar um **ritmo sustentável** de trabalho, evitando a prática de hora-extra.

Os requisitos do programa são colhidos durante o **jogo do planejamento**, quando o cliente escreve histórias que podem ser implementadas em uma iteração. Cientes de que o cliente nem sempre sabe exatamente o que ele deseja, e ciente de que clientes raramente podem negociar o tempo ou o custo de um projeto, durante esta atividade os programadores e os clientes negociam o escopo do software que será entregue construindo **metáforas** comuns para facilitar a comunicação. Os programadores são responsáveis por estimar quanto tempo de desenvolvimento cada história

necessita e os clientes por priorizar as histórias que serão entregues em uma iteração. Ao final de um *release*, uma reunião é realizada com os clientes com o intuito de repriorizar histórias.

O desenvolvimento de uma iteração se dá por equipes de dois programadores, que se alternam ao longo do tempo e com a **presença constante do cliente** e apoio do treinador. A equipe realiza reuniões diárias para se organizar. Os programadores escrevem código para funcionalidades que podem ser implementadas em períodos curtos de tempo, buscando sempre manter o *design* **simples**. O código é produzido de acordo com o **padrão de codificação** estabelecido pela equipe, é **integrado constantemente** e conta com uma cobertura de **testes automatizados**, de unidade e de aceitação, esses últimos escritos em colaboração com o cliente. Existe **propriedade coletiva do código**, qualquer parte do software pode ser alterada por qualquer par e **refatorações** constantes são efetuadas para manter o *design* da aplicação o mais simples possível. O espaço de trabalho deve ser configurado para permitir a **programação pareada** e valorizar a comunicação, tendo quadros brancos e espaço para o acompanhador colocar cartazes informativos.

As regras de XP não são absolutas, sabendo que cada empresa tem sua própria cultura, XP é ágil o suficiente para poder ser adaptada aos diferentes contextos, as práticas não precisam ser seguidas à risca (porém, algumas combinações, como refatorar sem testar, não são recomendadas) e nada impede que desvios ocorram eventualmente ou outros padrões sejam utilizados em conjunto com a metodologia.

## 2.2 XP - Por que funciona?

Ao redefinir a metodologia na segunda versão de seu livro, Beck se concentra nas razões que fazem XP funcionar. A nova definição revela a principal dificuldade em implantar a metodologia em qualquer contexto:

“Programação eXtrema (XP) trata de mudança social”

(BECK, 2004, p.1)

A dificuldade mais encontrada ao tentar adotar XP em uma organização é justamente efetuar esta mudança. Isto significa deixar para trás velhos hábitos e padrões e abandonar defesas que nos protegem, mas interferem na nossa produção, como o receio de mostrar código produzido para avaliação dos colegas. Beck sugere que o sucesso é fruto de boas relações humanas e alguma técnica.

Em XP, ser sincero sobre a capacidade de produção e então produzir aquilo que foi planejado, crescendo ao mesmo tempo que nos relacionamos com outros e escrevendo código de qualidade implica melhores relações humanas na sua organização e, conseqüentemente, bons negócios. XP alinha muita comunicação e trabalho em equipe com algumas técnicas de programação para garantir software de qualidade entregue no tempo certo.

A metodologia tem quatro componentes. Em primeiro lugar, uma filosofia com valores e princípios bem definidos. Uma organização disposta a mudar deve valorizar a comunicação, o *feedback*, a simplicidade, a coragem e o respeito.

Em segundo lugar, ferramentas que traduzem os valores em práticas concretas, pelas quais um grupo pode se responsabilizar. São os princípios de XP:

- humanidade
- economia
- benefício mútuo
- auto-semelhança
- melhoria
- diversidade
- reflexão
- fluxo
- oportunidade
- redundância
- falha
- qualidade
- pequenos passos
- responsabilidade

Em terceiro lugar, temos as práticas, que expressam os valores e seguem os princípios, complementando-se e amplificando-se:

- sentar juntos
- time completo
- área de trabalho informativa
- trabalho energizado
- programação pareada
- histórias
- ciclo semanal
- ciclo de estação
- folga
- *build* veloz
- integração contínua
- programação orientada por testes
- *design* incremental
- envolvimento real com o cliente
- implantação incremental
- continuidade do time
- redução do time
- análise da causa inicial
- código compartilhado
- código e testes
- repositório único de código
- implantação diária
- contrato de escopo negociável
- pague pelo uso

Em quarto lugar, temos uma comunidade que compartilha esses valores, princípios e práticas.

XP é diferente de métodos tradicionais em diversos pontos. Tem um ciclo de desenvolvimento curto baseado num planejamento incremental de escopo negociável. Além disso, o progresso é monitorado através da evolução de testes automatizados que garantem o funcionamento do software. A equipe prioriza a comunicação oral, testes e o código fonte para entender e trabalhar com o sistema. O processo de *design* é incremental, buscando manter o ritmo de entrega de software contínuo e a aplicação simples e funcional durante toda a vida do sistema. XP cria um ambiente de colaboração próxima e engajada, no qual as práticas funcionam de acordo tanto com os instintos de curto prazo da equipe, quanto nos interesses de longo prazo do projeto.

A nova descrição de XP diz que a metodologia é leve, para qualquer tamanho de time, dando enfoque às restrições do desenvolvimento de software e que se adapta a requisitos vagos ou mutantes, implicando que todos fazem XP de maneira diferente e com diferentes graus de sucesso. O único caso quando XP não se aplica é quando uma organização não consegue ou não pode mudar seus valores.

Aos programadores a mensagem é que XP funciona se você perceber que não é seu trabalho administrar as expectativas dos outros e sim fazer o seu melhor e comunicar-se com frequência e clareza, jogando para vencer e aceitando responsabilidade.

XP assume como pressuposto que as pessoas fazem parte de um time, querem trabalhar juntas e se aperfeiçoar, estão dispostas a mudar e que estas mudanças não custam nada, ou custam pouco.

XP funciona por que lida com os riscos do desenvolvimento de software. XP limita atrasos com implantação diária, ciclos semanais que oferecem *feedback* granular do progresso e sempre desenvolvendo a maior prioridade do cliente primeiro. XP evita o cancelamento do projeto ao sempre entregar o menor *release* possível com o maior valor agregado. XP evita que o sistema se torne defeituoso mantendo uma suíte de testes automáticos e o sistema sempre integrado, pronto para ser implantando. XP lida com defeitos tanto com testes de unidade quanto com testes de aceitação. XP lida com erros de negócio ao ter envolvimento real do cliente que aprende e evolui junto com o time completo. XP se adapta a mudanças de negócios com implantação diária e ao possibilitar mudanças no plano durante um ciclo de desenvolvimento. XP evita o excesso de funcionalidades que não trazem benefícios diretos ao sistema ao só implementar histórias de alta prioridade escritas pelo cliente. XP lida com a alta rotação de trabalhadores criando um ambiente de convívio agradável e com menos estresse. Ao trabalhar com estimativas feitas pelos próprios desenvolvedores, pelas quais eles se responsabilizam e ao incentivar o contato humano, evita-se evasão de membros da equipe devido a frustrações ou estresse. Ao incentivar novatos na equipe, XP acolhe novos membros com tranquilidade.

## 2.3 Valores

A seguir iremos apresentar os valores da Programação eXtrema e fazer uma breve análise de cada.

### 2.3.1 Comunicação

O desenvolvimento de software é uma atividade complexa. Por mais simples que seja um programa, no mínimo duas pessoas estão envolvidas em sua implementação: um programador e um cliente. Mesmo os sistemas mais simples requeridos pelo mercado normalmente exigem uma equipe de programadores trabalhando em conjunto. Por isso, valorizar a comunicação é muito importante. XP pretende manter o custo de tempo e energia para descobrir uma informação baixo e a taxa de dispersão da informação alta.

Sempre que mais do que uma pessoa está trabalhando em alguma tarefa, a eficiência e eficácia de um projeto está diretamente ligada à qualidade da comunicação entre estas pessoas. A grande maioria dos processos existentes valoriza a comunicação; o problema é que muitas metodologias acreditam que o problema de comunicar-se pode ser resolvido com documentação escrita, extensa e completa. XP inova ao priorizar a comunicação pessoal e oral, acreditando que falar é melhor do que escrever. Ao estar em contato presencial com uma pessoa, sinais sutis como a linguagem corporal

podem enriquecer em muito a comunicação, além disso, este contato permite que as dúvidas sejam resolvidas e discutidas logo que surgirem. Já a documentação escrita sempre tende a desatualizar-se rapidamente.

Estudos recentes [46] mostram que a comunicação verbal é mais eficiente que a comunicação escrita, pois ao escrever, uma pessoa corre o risco de assumir que certas sutilezas serão percebidas pelo leitor, devido a um fenômeno psicossocial bem estabelecido, o egocentrismo, que diz que pessoas tem dificuldades em se distanciar de suas próprias perspectivas e entender como serão interpretados por outros.

A comunicação é valorizada em XP de diversas maneiras. Encontros pessoais entre os desenvolvedores e o cliente acontecem freqüentemente, durante os jogos de planejamento e as reuniões de avaliação das iterações. Criar uma metáfora comum para facilitar a comunicação é uma das práticas sugeridas pela primeira versão da metodologia e defendida por Beck [17]. O cliente sempre presente permite que desenvolvedores resolvam questões sobre requisitos ou validem a implementação de funcionalidades imediatamente.

Os desenvolvedores devem trabalhar em um mesmo espaço, fazendo pequenos encontros diários, para planejar quais tarefas serão executadas por quais pares. Além disso, verifica-se o andamento do dia anterior.

O espaço de trabalho tem uma função importante na comunicação e é explorado na metodologia que prega o uso de cartazes informativos espalhados pelas paredes. Estes cartazes contêm métricas do projeto, são os chamados *radiadores de informação* [29]. Através destes cartazes, os desenvolvedores e até mesmo o cliente podem rapidamente absorver informações ligadas ao andamento do projeto. A qualidade do processo é avaliada constantemente e comunicada de maneira quase osmótica.

A programação pareada, com pares que se alternam ao longo do tempo, trabalhando sobre todo o código e integrando-o freqüentemente contribui para que o *design* e a implementação do software seja transmitido de maneira tácita por toda equipe.

Finalmente, o fato das equipes de XP serem pequenas (de no máximo 12 desenvolvedores, sendo 12 o limite natural de pessoas com as quais uma pessoa consegue manter comunicação confortavelmente durante um dia) [19], permite que a comunicação pessoal seja de fato eficiente. Equipes maiores necessitam de controles mais rígidos e comunicação mais estruturada. Veremos porém que XP pode escalar, isso será discutido na seção sobre a nova versão.

A comunicação é aquilo que tem mais importância em uma equipe de desenvolvedores, essencial para uma sensação de pertinência e cooperação efetiva.

### 2.3.2 Simplicidade

Os desenvolvedores de software conhecem a regra dos *80 por 20*, ela diz que 80% dos benefícios são fruto de 20% do trabalho. Justamente por essa razão que deve-se valorizar a simplicidade. Um sistema com muitas funcionalidades tem quase todo seu valor em 20% destas, manter o sistema simples e sempre implementar as histórias de maior prioridade do cliente evitam o fracasso. XP prega que manter o sistema simples é essencial para não gerar mais trabalho. Desenvolvedores não devem generalizar sem necessidade, ou supor necessidades, devem implementar da maneira mais simples possível os requisitos mais prioritários do cliente. Por ter autonomia de mudar qualquer parte do código, os programadores estão sempre atentos a oportunidades de refatorar o software com o objetivo de simplificá-lo.

Manter um sistema simples é uma atividade intelectual intensa, que requer esforço de todos e, justamente por isso, depende do contexto. Uma equipe com desenvolvedores experientes pode achar uma solução de *design* que usa padrões de projeto como *Observer* simples, enquanto que se a equipe tiver pessoas que não conheçam o padrão, esta solução deixa de ser simples.

Simplicidade e comunicação são valores complementares. Quanto mais simples o sistema, menos precisa ser comunicado e quanto maior a comunicação, maior o entendimento e portanto mais fácil a manutenção da simplicidade.

### 2.3.3 *Feedback*

*Feedback* é valorizado pois a equipe que faz XP é ágil. Para poder mudar o plano e se adaptar, precisa saber, rapidamente e com exatidão, o que está acontecendo. Ao longo do desenvolvimento, é muito importante receber *feedback* do cliente, a fim de avaliar se o software está de acordo com suas necessidades. Com *releases* rápidos e frequentes, ao ver o software funcional, o cliente pode entender o que ele realmente precisava, mudando de idéia, ou descobrir requisitos dos quais ele não estava ciente.

Durante os encontros de avaliação e planejamento, o cliente recebe *feedback* dos desenvolvedores quanto às implicações de implantar seus requisitos, incluindo informação sobre quanto tempo a implantação deve consumir. A equipe então recebe *feedback* sobre quais funcionalidades devem ser

priorizadas, podendo alterar o software a tempo de manter o valor para o cliente.

Programação pareada também permite que os programadores recebam *feedback* imediato de seus pares sobre o código que estão produzindo. Este processo de inspeção contínua comprovadamente [150, 93, 72] reduz a frequência de erros e aumenta a produtividade de ambos programadores.

Testes automatizados provêm *feedback* imediato sobre o funcionamento do software aos desenvolvedores. É este *feedback* que permite que refatorações do software sejam efetuadas com baixo risco. Além disso, muito *feedback* pode ser recebido através de ferramentas de desenvolvimento. Ao usar um ambiente de desenvolvimento integrado, como o Eclipse por exemplo, *feedback* sobre o código produzido é recebido em tempo real, erros de sintaxe são detectados e corrigidos imediatamente, quase automaticamente.

O acompanhador analisa e disponibiliza diferentes métricas sobre o desenvolvimento à equipe; este *feedback* contínuo permite que todos conheçam a realidade do projeto e do processo. Quando o desenvolvimento demora mais tempo do que estimado, a equipe fica sabendo disso e pode notificar o cliente a tempo, possibilitando que o escopo de um *release* seja renegociado.

Valorizar o *feedback* é ter satisfação em melhorar e saber que não existe perfeição instantânea, é também valorizar comunicação. Simplicidade também contribui com o *feedback*, quanto mais simples um sistema, mais fácil é obter *feedback* sobre seu *design* e implementação.

### 2.3.4 Coragem

Desenvolvimento de software nos dias de hoje é um processo que causa medo, tanto o cliente como os desenvolvedores têm muitos medos. XP lida com esses medos ao fornecer o suporte necessário para que as pessoas possam sentir coragem para agir e tomar decisões. As práticas de XP se apóiam, dando confiança à equipe. Os programadores podem ter coragem de refatorar, pois sabem que os testes irão detectar erros imediatamente. O cliente pode decidir com mais coragem, quando avalia o software funcional após cada *release*, sabendo que pode priorizar as funcionalidades que lhe são mais importantes. No jogo do planejamento, são os desenvolvedores que estimam as funcionalidades, podendo ter confiança de que irão entregar o que estão prometendo ao cliente a tempo. Ao ter controle do próprio processo, os desenvolvedores têm consciência da velocidade na qual trabalham, podendo encarar a necessidade de retrabalho com coragem.

Coragem pode ser perigosa se não balanceada com os outros valores. É fácil ter coragem de não documentar o código, porém se a comunicação não for valorizada, essa estratégia expõe a equipe a

altos riscos. Coragem para falar valoriza comunicação e confiança. Coragem de descartar código valoriza a simplicidade. Coragem de buscar respostas valoriza o *feedback*.

## 2.4 Práticas

As doze práticas de XP são simples, porém a riqueza da metodologia provém da combinação das práticas. Poucas práticas se sustentam de maneira individual, pode ser inclusive perigoso adotar uma prática sem que seus defeitos possam ser compensados pelas práticas que a apóiam. A Figura 2.1 mostra um grafo de relação entre as práticas:

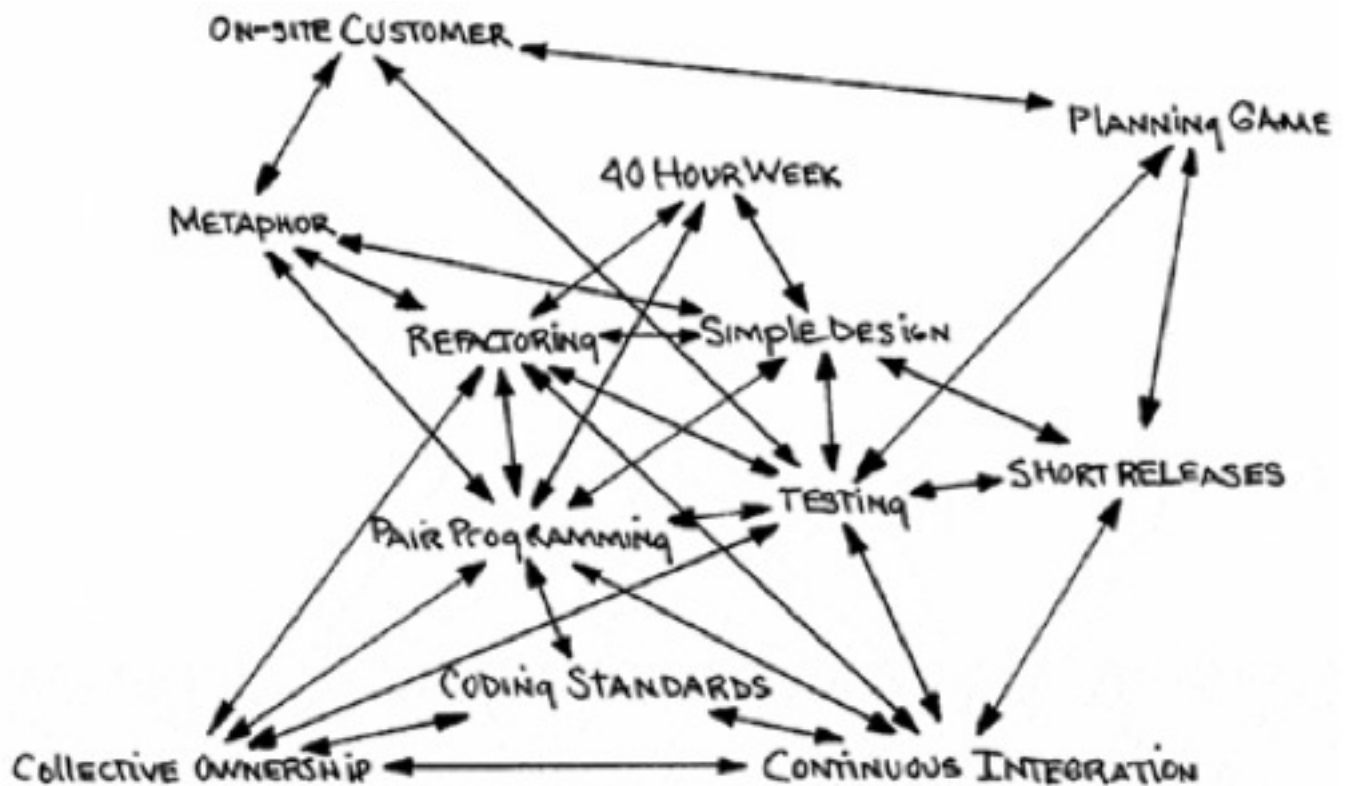


Figura 2.1: As práticas se apóiam (BECK, 1999)

A seguir iremos apresentar as doze práticas da primeira versão de XP, explicitando, para cada uma delas, as relações com as outras práticas já apresentadas.

### 2.4.1 Jogo do planejamento

Não importa quais precauções alguém pode tomar, seu plano nunca andará exatamente de acordo com o que foi planejado. O Jogo do Planejamento é a prática de sempre revisar o plano, com uma frequência constante de encontros nos quais os desenvolvedores e os clientes devem entender o que foi feito, o que não foi feito e o porquê, além de qual o novo plano a se seguir. É uma prática que descreve alguns processos, explicitando atividades a serem realizadas pela equipe regularmente.

Beck e Martin Fowler [80] discutem em detalhes como planejar um projeto XP, citando 3 razões para realizar o planejamento:

1. Precisamos ter certeza que sempre estamos trabalhando na coisa mais importante que temos para fazer.
2. Precisamos estar coordenados com outras pessoas.
3. Quando um evento inesperado ocorre, precisamos entender as conseqüências para as duas primeiras.

As atividades de planejamento do jogo do planejamento envolvem os programadores, o treinador, o acompanhador e o cliente, porém Beck [16] deixa claro que somente duas forças atuam nesse jogo: o desenvolvimento e o negócio em uma relação de respeito e benefício mútuo. Normalmente, o trabalho do treinador é ajudar o cliente a escrever histórias e intermediar o jogo com a equipe de desenvolvimento. O treinador apóia a equipe com dados relativos ao desempenho mais recente de todos membros dessa. É com base nesses dados que os desenvolvedores são capazes de estimar o tempo necessário para concluir uma história, Martin Fowler [52] nomeia este padrão de “clima de ontem”.

William Wake [155] descreve o jogo do planejamento como dois processos: o planejamento de um *release* e o planejamento das iterações deste *release*. O primeiro processo se dá em duas etapas: a etapa de exploração e a etapa de planejamento. Na etapa de exploração, o negócio é representado pelo cliente, que escreve e reescreve histórias, os programadores estimam histórias e escrevem *spikes*, pequenos protótipos que serão descartados. A atividade pode ser resumida da seguinte forma:

- O cliente escreve as histórias que gostaria de ver no *release*.

- Os programadores estimam o tempo necessário para concluir a história.
- Se a história for muito grande os desenvolvedores pedem ao cliente para rescrevê-la, quebrando-a em histórias menores.
- Se os desenvolvedores não sabem estimar a história, eles partem para o desenvolvimento de um *spike* para ter uma base sobre a qual podem estimar.

Ao final desta fase (que pode durar de alguns dias a uma semana) todas as histórias do *release* estão estimadas. Pode-se então iniciar a etapa do planejamento das iterações que compõem um *release*, que normalmente dura algumas horas. Com todas as histórias em mãos, o cliente avalia o valor de cada história, priorizando aquelas que dão o maior retorno ao seu negócio. O acompanhador então, usando dados da última iteração, declara a velocidade da equipe. Ou seja, baseando-se no desempenho passado, sabendo a estimativa das histórias e também o tempo real que elas levaram e comunicando estes dados à equipe, podemos ter uma idéia de quantas histórias poderão ser realizadas em uma nova iteração. O cliente então escolhe as histórias que devem compor o escopo do *release*. Somando suas estimativas e usando a velocidade calculada pelo acompanhador, sabemos dizer quantas iterações são necessárias para completar o *release*. O escopo pode ser negociado baseado nessas informações, o cliente pode mudar as histórias que deseja para obter o *release* mais cedo, por exemplo.

O segundo processo é o planejamento de uma iteração. Este processo ocorre ao começo de cada iteração e também é composto por duas fases, um *brainstorming* inicial e a fase de aceitação de tarefas. Durante a primeira fase os desenvolvedores, na presença do cliente, pegam as histórias planejadas para a iteração e escrevem tarefas concretas a serem realizadas. Cada desenvolvedor, com a ajuda do acompanhador, sabe quanto trabalho consegue realizar em uma iteração, durante a segunda fase esse conhecimento é importante, pois os desenvolvedores assumem tarefas e estimam o tempo necessário para completá-las. Enquanto para estimar histórias toda equipe participa, ao estimar uma tarefa é o desenvolvedor que a aceitou que deve dizer o tempo necessário. Este processo ocorre em paralelo, até todos os desenvolvedores terem uma carga de trabalho balanceada para toda a iteração.

Além destes dois processos, o jogo do planejamento descreve mais duas atividades, a validação de uma iteração e, opcionalmente, o replanejamento de um *release*. Ao final de cada iteração o cliente deve se encontrar com os desenvolvedores para validar o código escrito, rodando todos os testes funcionais das histórias previstas na iteração. O cliente tem a opção de aceitar os resultados da iteração ou não, os desenvolvedores podem explicar por que estimativas estouraram (se isso de

fato ocorrer) e avisá-lo se algum problema inesperado foi detectado. Dependendo do resultado deste encontro o cliente tem a oportunidade de replanejar o *release* atual. Se necessário, o cliente deve repriorizar histórias marcadas para as próximas iterações, resgatar histórias que não foram validadas para que continuem sendo desenvolvidas, ou até mesmo criar novas histórias. É importante notar que estimativas e prioridades de histórias não são absolutas, o cliente deve tentar adiar as decisões o máximo possível (mas não mais tarde do que isso) para baseá-las na melhor informação disponível.

O jogo do planejamento pode sofrer variações, adaptando-se melhor ao contexto cultural da organização onde XP está sendo aplicada. Esta prática valoriza diretamente o *feedback*, pois ocorre com frequência, dando mais oportunidades para o cliente mudar de plano, além de usar métricas de iterações anteriores e *spikes* para melhorar o processo de estimativa. Ao ditar que os desenvolvedores são donos das estimativas, a prática valoriza a coragem. A equipe sabe que prometeu ao cliente exatamente aquilo que pode cumprir. A comunicação é valorizada durante todo processo de exploração, no qual a presença contínua do cliente e seu diálogo com os desenvolvedores é essencial para construir estimativas reais e para que a equipe entenda muito bem os requisitos do cliente.

#### 2.4.2 *Releases* pequenos

A prática de *releases* pequenos define o ritmo dos eventos que marcam o progresso em um projeto XP. O progresso se dá pela entrega de software funcional, um *release* que deverá ser colocado em produção pelo cliente. A entrega deve agregar o máximo de valor para o cliente com a menor quantidade possível de código novo. O ciclo entre entregas deve ser frequente e o mais curto possível, sendo a única restrição a de que o sistema entregue deve fazer sentido como um todo.

*Releases* pequenos aumentam o *feedback* que a equipe recebe quando o cliente avalia a produção que foi planejada. São também *feedback* direto para o cliente, que vê quanto progresso foi feito e pode usar informações recentes para mudar prioridades ou até mesmo criar histórias novas para o próximo *release*. É importante que o cliente use de fato a nova versão do software após cada *release*.

Além de valorizar o *feedback*, esta prática também valoriza a comunicação, pois o sistema funcional comunica ao cliente exatamente quais requisitos foram implementados. Valoriza também a coragem, pois ao estabelecer um ritmo de entregas tanto o cliente sente mais coragem com a evolução incremental do sistema, quanto os desenvolvedores sentem coragem sabendo que irão manter um ritmo de entregas funcionais. Sendo pequenos e contendo somente as funcionalidades de maior prioridade, os *releases* pequenos também valorizam a simplicidade.

Esta prática é apoiada pelo jogo do planejamento, que ajuda a priorizar as histórias de mais valor, fazendo com que até mesmo um sistema pequeno tenha valor para o negócio.

### 2.4.3 Metáfora

Metáfora é a prática que prega que a aplicação desenvolvida possua um “sistema de nomes” mapeando os elementos do sistema a nomes de objetos do mundo real. A metáfora facilita a comunicação entre os desenvolvedores e também facilita a comunicação dos desenvolvedores com o cliente. Ela descreve uma atividade que deve ser exercida em uma relação entre os papéis do cliente e dos programadores.

Um bom exemplo de metáfora é a usada para o software de folha de pagamento da Chrysler, no qual os componentes do sistema eram nomeados como componentes e partes de uma fábrica. Essa escolha funcionou bem pois, além dos desenvolvedores, outras pessoas na organização entendiam bem os conceitos de uma fábrica (a Chrysler é uma fábrica de automóveis). O sistema de nomes continha “peças” como a “peça salário”, que seguiam em diferentes “linhas de montagem”, transportadas em “baldes” e processadas em diferentes “estações”, calcular a folha de pagamento não é algo que pode ser trivialmente comparado a uma fábrica, mas esta metáfora facilitou para que tanto os desenvolvedores, quanto o cliente, entendessem o sistema em toda sua complexidade.

A metáfora ajuda todos os envolvidos a terem uma visão comum do sistema, seus elementos, como eles funcionam e como poderiam funcionar. Também provê um vocabulário comum. Ao se referir à metáfora, tanto o cliente, quanto os desenvolvedores, podem entender o que o outro quer comunicar. Não importa se o cliente está pensando em uma regra de negócio enquanto os desenvolvedores estão pensando em um objeto do sistema. Desta maneira a metáfora valoriza a comunicação. A metáfora também ajuda a explorar os limites do sistema, valorizando o *feedback*, pois ao explorar a metáfora, o time pode descobrir fatos que não sabia sobre o sistema, ou pode ter idéias novas.

Escolher uma metáfora boa é um trabalho contínuo, que começa na fase de exploração, é usada para guiar o *design* e para entender como classes de objetos se relacionam, é importante que o time todo concorde com a metáfora e que ela faça parte da comunicação diária.

A metáfora também valoriza a simplicidade ao tentar explicar de maneira fácil um sistema que pode ser complexo e ao facilitar o entendimento do sistema como um todo tanto pelos desenvolvedores quanto pelo cliente.

#### 2.4.4 *Design* simples

Faça a coisa mais simples possível. Esta é a estratégia principal da prática de *design* simples que prega que o programador deve sempre se esforçar para manter o design da aplicação simples e ao mesmo tempo evitar complexidade ou flexibilização desnecessária até o momento em que esta se torne crucial. O *design* mais simples a qualquer momento é aquele que roda todos os testes, não contém lógica duplicada, esclarece todas as intenções aos programadores e tem o menor número possível de classes, métodos e linhas de código. É uma prática que determina um processo contínuo de avaliação da qualidade do código fonte gerado pelo desenvolvimento.

Se o *design* é simples, adicionar funcionalidades também será simples. Ao evitar código duplicado, quando surge a necessidade de adicionar algo, será necessário alterar somente uma parte do programa e será fácil descobrir isso se o *design* demonstra claramente todas as intenções aos programadores. O *design* simples facilita localizar onde e perceber como, deveremos alterar o programa. Ao mesmo tempo, ao rodar todos os testes com o mínimo de classes e métodos, garantimos que não existe desperdício ou redundância negativa no sistema.

Esta prática valoriza, obviamente, a simplicidade e, também, a comunicação e o *feedback*, pois o *design* simples comunica de maneira mais fácil as suas intenções aos programadores. Ao manter a simplicidade, valorizamos também a coragem, pois mudanças podem ser feitas com mais facilidade.

*Design* simples é uma prática que corrobora *releases* pequenos, é a simplicidade do *design* que garante que ao ter uma aplicação que faz só o suficiente para uma entrega, as entregas podem ser realizadas com mais frequência. Ao mesmo tempo, a prática da metáfora possibilita que mudanças no *design* se mantenham consistentes e que o *design* da aplicação como um todo tenda a convergir e manter-se simples.

#### 2.4.5 Testes automatizados

Para garantir a qualidade do software construído é essencial a presença de testes automatizados que validem o seu funcionamento. Estes testes aumentam a confiança tanto dos programadores quanto do cliente de que o sistema funciona de fato. Esta prática define que testes são artefatos produzidos por uma equipe XP como atividade diárias, tanto na interação entre programadores, quanto na interação com o cliente.

Esta prática incentiva a construção de testes de unidade automatizados, que testam cada com-

ponente do sistema individualmente, sendo escritos pelos programadores para cada funcionalidade dos componentes. Beck sugere que o desenvolvimento seja orientado por testes [18], programadores devem primeiro escrever um teste que falha para qualquer funcionalidade, depois fazê-lo passar, refatorando o código para remover duplicação, repetindo este ciclo até ter todas as funcionalidades testadas. A todo momento, todos testes de unidade devem passar.

Testes de funcionalidade, também conhecidos como testes de aceitação, fazem parte desta prática, estes validam o sistema como um todo do ponto de vista das necessidades do cliente e, de preferência, devem ser escritos pelo próprio cliente com a ajuda de um desenvolvedor. [Os testes de funcionalidade aumentam a probabilidade que os requisitos do cliente estão sendo cumpridos e são uma ferramenta para que o cliente acompanhe o progresso do projeto.](#) Quando todos testes de funcionalidade passarem, o sistema está pronto. Testes de funcionalidade também devem ser automatizados.

Esta prática valoriza o *feedback*, pois os testes mostram o estado atual de funcionamento do sistema. [Valoriza também a coragem, pois ao passar, os testes diminuem a probabilidade de erros \[92,62\], dando mais segurança à equipe de que o sistema está funcionando.](#)

O *design* simples facilita a escrita dos testes: se o código evita acoplamento e mantém alto nível de coesão, criar testes de unidade é fácil. O fato de testes serem feitos durante o desenvolvimento e não em uma fase final, ajuda na prática de manter os *releases* pequenos.

#### 2.4.6 Refatoração

Refatoração [50] é a técnica de melhorar algum aspecto não funcional do código fonte de um programa. O objetivo é criar código mais simples de ler e entender, melhorando o *design* do código. A prática de refatoração contínua incentiva os programadores a ficarem atentos a oportunidades de melhorar o software através da refatoração, definindo um processo contínuo. Essas oportunidades ocorrem quando deve-se adicionar algo novo ao programa, antes de implementar um novo requisito, verificamos se existem oportunidades de simplificar o *design* para facilitar esta implementação. Após adicionar o requisito, também temos a oportunidade de refatorar o código e torná-lo ainda mais simples. A refatoração contínua valoriza a simplicidade e a comunicação.

Existem catálogos de refatorações [50,81] que listam refatorações comuns que podem ser aplicadas no dia-a-dia, apresentando motivações para cada refatoração, uma mecânica que lista pequenos passos que devem ser seguidos para executar a refatoração e exemplos práticos. Um bom programador se acostuma a identificar sintomas de que o código precisa ser refatorado, como classes muito grandes,

métodos muito grandes, comentários demais e, principalmente, código duplicado. Algumas refatorações são simples e implicam somente em mudança de nomes de variáveis, parâmetros ou classes, outras implicam a aplicação de padrões de *design* orientado a objetos [61]. Existem ainda técnicas para refatorar bancos de dados [6]. É altamente recomendável o uso de ferramentas de refatoração, que facilitam a adoção da prática automatizando a mecânica da maioria das refatorações.

A prática de refatoração enfatiza a evolução da metáfora, ao refatorar continuamente, refina-se o entendimento do que a metáfora significa na prática.

Manter o design simples é possível com o apoio da prática de refatoração. Ao usar técnicas de refatoração conseguimos mudar o *design*, simplificando-o sem os medos e as preocupações normalmente associadas a esta atividade. Ao mesmo tempo, o fato do *design* ser simples, facilita refatorações.

A prática de testes automatizados é essencial para a refatoração, os testes são a garantia que a refatoração não afetou o funcionamento do programa.

#### 2.4.7 Programação pareada

A prática de programação pareada demanda que todo código fonte que será entregue ao cliente, seja produzido ao mesmo tempo, por duas pessoas. [Inicialmente dizia-se que o par deveria usar o mesmo computador, porém variantes surgiram nos quais o par pode usar uma só máquina, usar uma máquina com dois teclados e mouses, compartilhar a mesma sessão de trabalho em duas máquinas diferentes, podendo estar co-locados ou até mesmo trabalhando a distância.](#) Uma equipe de Programação eXtrema é sempre composta por um grupo de pares, cada par é mais produtivo e cria mais código de maior qualidade, do que se as pessoas programassem individualmente [93]. Programação pareada define como os programadores devem interagir diariamente, a eficácia desta técnica já foi comprovada até mesmo em contextos que não utilizam as outras práticas de XP [28].

Ao iniciar uma sessão de programação pareada, o par deve discutir as tarefas que serão executadas relativas à história que estão implementando. Após chegar a uma estratégia comum, o par alterna de papéis durante o tempo em que trabalha junto. Uma das pessoas assume o papel de “motorista”, controlando o teclado e o *mouse* e concentrando-se no código que será escrito. A outra pessoa assume o papel de “chapa” e ajuda o motorista no seu trabalho, pensando sobre todas implicações do código que está sendo escrito de maneira mais estratégica, sugerindo testes e ficando atento a pequenos erros que o motorista talvez não perceba. Esta atividade é intensa para ambos os papéis e a comunicação é importante para que seja executada com harmonia, aproveitando ao máximo o empenho de ambos

indivíduos. Ao longo de uma iteração, os pares são criados entre todos membros da equipe, isso implica que mais pessoas entendem o sistema de maneira mais completa do que se elas trabalhassem individualmente.

Programação pareada valoriza principalmente a comunicação, especialmente ao fazer um rodízio de pares, alternando pessoas entre os pares com certa frequência, para que o conhecimento sobre as diferentes partes do sistema seja compartilhado entre todos membros da equipe e a diversidade do time seja potencializada. Esta prática também valoriza a coragem, pois duas pessoas tem mais coragem do que uma e valoriza o *feedback* através do diálogo ativo entre o par.

Programação pareada é uma prática que influencia o *design* simples, duas pessoas trabalhando no mesmo problema aumentam as chances de que o *design* seja de fato simples e não simplório. Ao mesmo tempo, a simplicidade garante que ambos possam compreender o que está acontecendo durante a implementação.

Esta prática também apóia a prática de testes, ao responsabilizar o chapa a pensar em novas idéias para testes que podem ser feitos no código sendo escrito pelo motorista. Ao mesmo tempo, a prática de pensar em testes em conjunto ajuda ambos a alinhar seu entendimento do problema antes de ter que resolvê-lo. A programação pareada também ocorre quando um programador ajuda o cliente a escrever testes de aceitação.

Ao parear, uma dupla sente mais coragem para refatorar e, ao mesmo tempo, com a redundância de duas pessoas concentrando-se no mesmo código, diminui a probabilidade de que alguma parte do sistema seja quebrada durante refatorações.

A prática da metáfora também ajuda na programação pareada, fazendo com que seja mais fácil para uma dupla chegar a um acordo de como desenhar o sistema e como nomear os objetos criados.

#### 2.4.8 Propriedade coletiva do código

Esta prática implica que todo o código fonte pertence a todos programadores da equipe e qualquer um tem a liberdade de alterar qualquer parte dele, mesmo se não foi o seu autor original. Além disso, pares devem estar atentos a oportunidades de melhoria do código e executá-las sempre que forem detectadas. Esta prática define mais uma maneira de como desenvolvedores interagem.

Como alternativa a outros modelos de propriedade de código, a propriedade coletiva valoriza a comunicação e a colaboração. Esta prática incentiva o compartilhamento de conhecimento e, ao

mesmo tempo, reduz o risco de alguma parte do software depender exclusivamente da pessoa que o criou. Ela também promove maior qualidade e agilidade, pois toda equipe conhece todo o código e pode mudá-lo sem hesitar quando detecta uma oportunidade para melhoria ou percebe a necessidade de alterações.

Ao praticar propriedade coletiva do código a equipe toda obtém algum conhecimento sobre o sistema por inteiro, ao mesmo tempo em que alguns tem mais conhecimento sobre as partes específicas com que trabalharam. O contato com todo o código fonte do sistema que ocorre graças a esta prática é *feedback* que os desenvolvedores obtém sobre o design e a arquitetura da aplicação, desta maneira esta prática valoriza o *feedback*. A prática também valoriza a coragem e a simplicidade ao incentivar todos a mudarem o código quando ele pode ser mais simples.

Ao praticar a refatoração os desenvolvedores devem estar atentos a qualquer melhoria que possa ser feita no código fonte, a prática de propriedade coletiva do código dá coragem aos pares que identificam a necessidade de mudanças em código que não tinham escrito originalmente. Além de apoiar a refatoração, esta prática corrobora a programação pareada ao diminuir possíveis disputas entre pares relacionadas à propriedade de um pedaço de código. Ao mesmo tempo, ao parear, programadores conseguem aprender mais rápido partes desconhecidas do sistema e também como alterá-las de maneira proveitosa. Duas pessoas atentas às modificações diminuem a chance de algo quebrar durante alterações.

Testes automatizados também enfatizam esta prática, pois aumentam a probabilidade de que eventuais mudanças não alteram o funcionamento do sistema.

#### 2.4.9 Integração contínua

Esta prática tem como objetivo garantir que todo código produzido será integrado a um repositório comum com a maior frequência possível, após algumas horas de desenvolvimento, ou um dia no máximo. Desta maneira, criar uma versão do software com todas as mudanças recentes e encaminhá-la ao cliente torna-se uma atividade simples e que pode ser inclusive automatizada.

Ao integrar, é importante verificar se não existem conflitos e que todos os testes de unidade estão passando. Garante-se desta maneira que as novas funcionalidades não tiveram impacto negativo no resto do sistema. Ao mesmo tempo, fica claro de quem é a responsabilidade de fazer o sistema funcionar, pois se os testes não passam a causa é obviamente o código que acabou de ser integrado. Testes automatizados apóiam e facilitam a prática de integração contínua.

Esta prática pode ser aplicada de diversas maneiras; o importante é manter o processo de *build* do sistema rápido e automático para que a integração possa ser compilada e os testes todos executados sem muita perda de tempo. Algumas equipes deixam uma máquina separada para integração, isto ajuda a tornar o processo seqüencial, evitando conflitos.

A habilidade de integrar o sistema a qualquer momento corrobora a prática de *releases* pequenos. Sempre existe uma cópia do sistema com todas as últimas funcionalidades que foram implementadas e esta cópia funciona e pode ser entregue ao cliente se necessário.

Ao integrar recebemos *feedback* sobre as mudanças que estamos incorporando ao código. Percebemos se uma refatoração teve implicações no trabalho de outros se, ao rodar testes depois da integração, alguma coisa quebra. Quando modificamos código devido a propriedade coletiva do código, se quebramos alguma coisa distante no sistema iremos ficar sabendo disso no momento da integração. Destas maneiras, a integração contínua apóia as práticas de refatoração, propriedade coletiva do código e testes, evitando que elas introduzam erros no sistema que passem despercebidos por muito tempo.

O *design* simples, mantido através de refatorações, apóia a integração contínua, pois fica mais fácil integrar novas mudanças e essas mudanças se mantêm simples e pequenas. Programação pareada também enfatiza esta prática, ao diminuir consideravelmente a quantidade de código que precisa ser integrada.

#### 2.4.10 Ritmo sustentável

Ritmo sustentável é uma das práticas que encontram mais resistência por parte de gerentes, talvez por ser inicialmente chamada de “semana de 40 horas”. O objetivo desta prática é evidenciar o fato de que não adianta desgastar sua equipe com trabalhos em hora-extra, pois a produtividade irá cair inevitavelmente. Seguindo um ritmo sustentável de trabalho, toda equipe estará bem disposta e motivada para trabalhar de maneira mais produtiva.

Ao mudar o nome desta prática Kent Beck atenta ao fato que não devemos exigir *exatamente* 40 horas de trabalho semanais, mesmo porque cada indivíduo tem um ritmo de trabalho diferente. O importante é manter todos descansados para evitar erros relacionados ao estresse e à sobrecarga. Existe a possibilidade da equipe trabalhar horas extras quando necessário, desde que isso não se torne rotina e que a equipe tenha a oportunidade de descansar após a época de sobrecarga.

Estar descansado implica em mais concentração e produtividade, esta prática apóia a refatoração,

pois diminui a probabilidade de erros, dando mais tranquilidade e coragem aos programadores. Corrobora também a prática de programação pareada, diminuindo a chance de brigas entre pares devidas a sobrecarga e estresse. Diminui também a probabilidade de conflitos entre a equipe toda, facilitando a propriedade coletiva do código.

O jogo do planejamento enfatiza esta prática, garantindo que a equipe tem uma quantidade razoável de trabalho a realizar e que ela poderá ser realizada no tempo disponível. Se algum problema ocorrer, replanejar é sempre possível, evitando assim a necessidade recorrente de trabalhar horas extras. Aliada à prática de testes automatizados, reduzimos também a frequência de surpresas ou dificuldades inesperadas e aumentamos a chance de conseguir manter um ritmo sustentável como planejado.

A própria prática do ritmo sustentável se apóia, por garantir que todos estão programando na melhor velocidade possível, mantendo um nível de produtividade e qualidade aceitáveis.

#### 2.4.11 Cliente sempre presente

Em XP, comunicação é essencial. Para melhorar a qualidade da comunicação entre os desenvolvedores e o cliente existe a prática do cliente sempre presente. Um cliente real, ou seja, alguém que conhece as necessidades daqueles que irão utilizar o sistema que está sendo construído, deve estar fisicamente junto da equipe de desenvolvedores. Desta maneira, ele poderá esclarecer dúvidas, especificar detalhes de histórias que estão sendo implementadas e, até mesmo, repensar o escopo de um *release* do projeto.

Esta também é uma prática difícil de adotar, principalmente pela relutância de gerentes de ter que ceder um funcionário para, além de executar o seu trabalho, acompanhar o desenvolvimento. Porém, a prática agrega muitas vantagens ao desenvolvimento, fazendo com que o sistema entregue seja de fato valioso para a organização e diminuindo muito os custos de tempo necessário para detalhar funcionalidades requeridas no programa.

Ao exigir a presença física do cliente, XP valoriza a comunicação. A maior parte das dúvidas que surgem durante o desenvolvimento podem ser respondidas imediatamente e através de comunicação verbal, deixando pouco espaço para que os desenvolvedores tenham que adivinhar o que o cliente realmente queria. Além disso, o cliente pode dar *feedback* instantâneo de histórias que estão sendo implementadas, podendo inclusive entender melhor o requisito que o seu sistema tem de fato. A prática elimina a necessidade de retrabalho que é freqüente quando desenvolvedores tomam decisões

de pequena escala equivocadas, delegando esta responsabilidade a um representante legítimo do negócio.

Além de responder dúvidas e como já vimos, participar ativamente do jogo do planejamento, o papel do cliente é também escrever testes funcionais automatizados. Para isso, o cliente deve primeiro especificar os testes das histórias que compõem um *release*, e então parear com algum desenvolvedor para codificá-los. Estes testes poderão ser executados de maneira automática por todos, complementando desta maneira a prática de testes automatizados.

Esta prática complementa também o jogo do planejamento, adiando a especificação detalhada de histórias para o melhor momento possível, que é quando elas estão sendo codificadas. Além disso, ao estar presente, o cliente tem acesso direto a informações sobre o andamento do projeto, podendo acompanhar todas as métricas organizadas pelo acompanhador e ser informado no momento certo se há necessidade de renegociar o escopo de um *release*. Além disso, ao estar presente, o cliente contribui para criação de uma metáfora consistente, apoiando esta prática também.

#### 2.4.12 Padrão de codificação

Esta última prática determina que no início do projeto, todos desenvolvedores da equipe concorrem sobre um padrão de estilo a ser usado no código fonte do sistema. Esta prática facilita a leitura do código por qualquer desenvolvedor e garante uma nomenclatura consistente em todo o sistema, independente do programador que codificou determinada parte. Com o passar do tempo, fica praticamente impossível dizer quem da equipe escreveu determinado pedaço de código, pois todos usam o mesmo estilo. Este padrão deve priorizar a comunicação, ser simples e minimizar a quantidade de trabalho necessário para adotá-lo. Além disso, ele deve ser aceito de maneira voluntária por toda a equipe.

Esta prática complementa muito bem a propriedade coletiva do código pois é fácil e natural mudar o código dos outros se ele é similar ao código que você escreve, não importa quem o criou originalmente. Além disso não existem brigas na equipe sobre qual estilo adotar, diminuindo também a possibilidade de atritos durante a programação pareada.

Padrão de codificação enfatiza a prática de refatoração pois facilita a alteração do código: refatorar não vai implicar em mudar também a formatação do código. Programação pareada também é mais fácil quando existe um padrão, pois o par não precisa discutir questões de formatação e pode se concentrar no problema a ser resolvido.

A prática da metáfora corrobora o padrão de codificação, pois além de usar as mesmas regras para criação de nomes de classes e métodos, a metáfora garante uma consistência de linguagem que facilita o entendimento do sistema e faz parte do padrão.

## 2.5 Papéis

XP define quatro papéis principais: os programadores, o cliente, o treinador e o acompanhador. Na primeira versão de XP, Beck [16] também cita 3 papéis secundários: testador, consultor e o “chefão”. Ao discutir as práticas, praticamente esgotamos as atividades e interações entre os programadores e o cliente. O papel de testador é absorvido pelos programadores, especialmente o programador que deve parrear com o cliente para escrever testes funcionais. O consultor é um papel externo, que surge quando há necessidade de experiência técnica, ele trabalha exatamente como um programador, pareando com o resto da equipe para disseminar seu conhecimento. O “chefão” tem o papel de patrocinar a equipe que está adotando XP e dar respaldo a ela dentro da organização. Resta explicitar melhor as atividades relacionadas aos papéis do treinador e do acompanhador, o que faremos a seguir.

### 2.5.1 Treinador

O treinador é uma pessoa com perspectiva independente, alinhado com os valores da organização, mas principalmente alinhado com os valores de XP. É experiente tanto na aplicação da metodologia e nas suas práticas, quanto em tecnologias. Deve ser um bom programador e conhecer práticas alternativas e outras metodologias ágeis, atuando para garantir a qualidade do trabalho. Este papel deve ser exercido por uma pessoa madura, que respeita a equipe e é respeitado por ela, sabendo falar quando necessário, mas principalmente sabendo ouvir.

O principal papel do treinador é ajudar a equipe a usar e entender XP durante o início do processo de adoção da metodologia por uma organização. XP é uma metodologia que requer alta disciplina, o treinador reforça esta disciplina, mantendo a equipe no processo correto. Não é recomendável adotar XP sem ajuda de um treinador experiente. Ao ajudar na implantação de XP por uma equipe, ele deve sugerir caminhos para adoção de práticas e estar atento ao andamento da equipe, ajudando-a a seguir a trilha correta. Por ser experiente, ele também atua como mentor de tecnologias e práticas como testes, refatoração e integração contínua. Sem um treinador de atitude pró-ativa que assume a liderança, a equipe fica a deriva na transição para Programação eXtrema, consequentemente a qualidade do trabalho diminui com o tempo e o risco de reverter a velhos hábitos e padrões aumenta.

O trabalho do treinador é presencial e deve ser feito, se possível, de maneira indireta. É mais importante fazer com que a equipe perceba um erro e descubra como resolvê-lo sozinha, do que resolver o erro. Seu papel é fazer com que as pessoas percebam o erro e não explicitá-lo. Um exemplo seria criar um teste que evidencia uma necessidade de simplificação do *design* em vez de dizer que o mesmo precisa ser simplificado. Ele dá liberdade para a equipe explorar, mas também avisa quando estão no caminho errado.

Durante seu dia-a-dia, o treinador exerce diversas atividades. Ele monitora o processo, chamando atenção da equipe quando as práticas não estão sendo seguidas. Ele reforça o processo, chamando a atenção de pessoas que não estão fazendo o seu melhor, ou até mesmo indo contra as práticas e, eventualmente, sugere que estas pessoas deixem a equipe. Ele muda o processo, adaptando práticas ao contexto local, ou até sugerindo novas práticas. Ele atua como mentor, pareando na programação em tarefas mais complexas ou em sessões de *design*. Ele também cuida da atmosfera relaxada da equipe e é responsável por garantir que o ritmo esteja sustentável.

Além de exercer estas tarefas, o treinador está atento a problemas que podem ocorrer. Se a velocidade de uma equipe cair, ele deve procurar a causa, reforçando práticas que podem retomar o ritmo, ou chamando o cliente para reduzir o escopo da iteração. Se o código está feio ou apresenta muitas duplicações, ele deve reforçar a prática de refatoração. Se a qualidade do software cair, ele deve reforçar a prática de testes.

O papel do treinador diminui de importância com o passar do tempo. Os próprios programadores, ao adquirir experiência com a metodologia, assumem o papel de monitorar o processo. Quando isso acontecer, o treinador pode deixar a equipe para que ela continue de maneira independente.

### 2.5.2 Acompanhador

O acompanhador é a consciência da equipe, ajudando o time a entender o progresso do desenvolvimento. Seu papel não é de dedicação exclusiva, exigindo pouco tempo de trabalho durante o dia-a-dia. Seu trabalho consiste em coletar informações para criar métricas do andamento do processo. Uma das principais atividades é coletar *feedback* das estimativas dos programadores, desta maneira medindo a velocidade do desenvolvimento.

Com estas informações, o acompanhador consegue dizer se a equipe vai conseguir terminar uma iteração entregando todas histórias prometidas, ou se é preciso renegociar o escopo com o cliente. Algumas informações básicas sobre o progresso do projeto que o acompanhador pode acompanhar

são relativas ao planejamento do *release*, como o número de histórias prometidas e as que já foram implementadas. O planejamento da iteração pode ser acompanhado através do número de tarefas selecionados para iteração em comparação as tarefas que já foram completas. O progresso de uma iteração pode ser medido observando o número de testes de aceitação criados e os que estão passando.

Além destas informações, métricas podem surgir para analisar informações que são valiosas para a equipe, que preocupam a equipe, ou que a equipe quer que outros saibam. É importante ter métricas negativas, principalmente quando precisamos evidenciar comportamentos problemáticos, que se repetem e que podem melhorar, mas é importante também ter métricas positivas. Ron Jeffries [76] sugere uma série de métricas e diferentes maneiras de visualizá-las, enfatizando que as métricas precisam estar visíveis para todos envolvidos, em gráficos grandes (e feitos à mão) valorizando a comunicação. É importante criar um espaço informativo, onde gráficos sobre as métricas mostrem de maneira objetiva o andamento do projeto. Estas informações são efetivamente o histórico do projeto. A Figura 2.2 é um exemplo de um gráfico que mostra o andamento de um projeto:

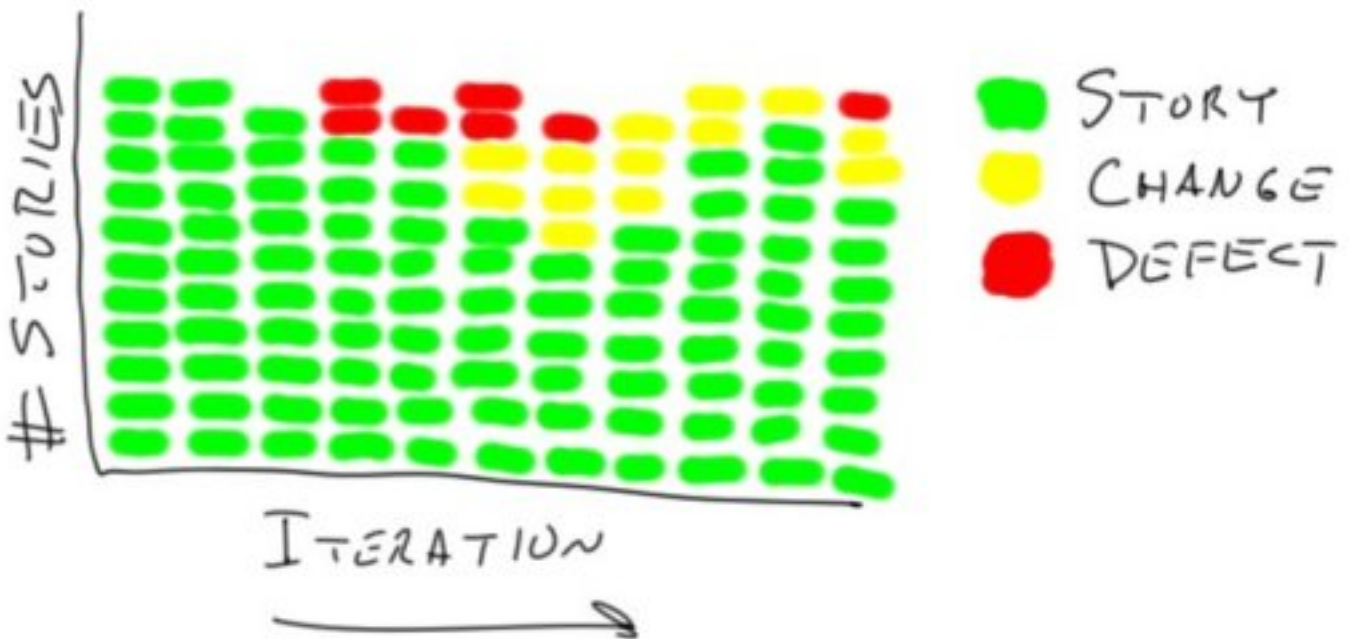


Figura 2.2: Histórias implementadas (em verde), mudanças em histórias (em amarelo) e correção de defeitos (em vermelho) realizadas em cada iteração (cada coluna é uma iteração). (JEFFRIES, 2004)

Durante o seu trabalho, o acompanhador pode usar diferentes suportes para suas métricas, como cartazes nas paredes, desenhos em quadros brancos, ou até mesmo planilhas. Dialogar com o treinador é valioso, pois ele pode sugerir métricas que irão evidenciar para a equipe possíveis problemas ou progressos importantes. Ao coletar métricas, o acompanhador deve tentar não incomodar a equipe. Além disso, aposentar métricas quando elas tiverem surtido o efeito necessário diminui a quantidade de informação visual no espaço, melhorando a comunicação e mantendo em evidência aquilo que no momento é importante para a equipe.

## 2.6 Refatorações e a adoção de novas práticas

Ao adotar XP, uma organização deve estar ciente de que o processo de chegar ao amadurecimento da metodologia é relativamente lento e trabalhoso. XP é uma metodologia que requer muita disciplina e que requer mudança social. Adotar seus valores leva tempo e implantar todas suas práticas também. Durante este processo, é importante adaptar XP ao contexto local, possivelmente refatorando algumas práticas, ou até mesmo criando novas práticas, ou pegando-as emprestadas de outros métodos ágeis. A seguir, iremos citar algumas possibilidades que tem nos ajudado a implantar XP em diferentes organizações.

Uma das práticas de Scrum [136] pode incrementar XP de maneira bem eficiente; é a prática de fazer encontros diários para micro-planejamento e comunicação da equipe. Estes encontros são chamados de *stand-up meetings* ou *papo-em-pé* na nossa tradução, por serem feitos com a equipe toda em pé, afim de não alongar as discussões. Nestes encontros a equipe descreve no que tem trabalhado, o que está funcionando bem para eles e dificuldades com as quais possam precisar de ajuda. Temos adotado essa prática pois ela garante que o encontro mantém-se curto e maximiza a comunicação efetiva, seguindo o princípio de respeito mútuo. Além disso, ao implantar XP, esta prática ajuda a evidenciar as dificuldades individuais e compartilhar soluções descobertas pelos membros da equipe [39].

O uso de ferramentas que auxiliam no processo pode ser determinante ao diminuir a curva de aprendizado de certas práticas. Em nossa experiência, um ambiente de desenvolvimento como o Eclipse ajuda no processo de implantação de XP ao auxiliar diretamente diversas práticas. Neste ambiente, padrões de codificação podem ser impostos diretamente no software e, ainda melhor, eles podem ser criados em conjunto no início de um projeto e compartilhados para todos desenvolvedores de maneira simples. Além disso, as facilidades de correção de sintaxe e até mesmo de auto-complemento de nomes de classes, variáveis e métodos fazem com que o Eclipse possa ser considerado

um ajudante extra para a dupla que está programando. Erros de sintaxe que escapem a atenção do chapa não passarão despercebidos por esta ferramenta. Quando ninguém da dupla consegue lembrar a assinatura de um método, a ferramenta faz isso para eles de maneira fácil e veloz. A fácil integração de ferramentas de testes ao ambiente torna o aprendizado e adoção da prática de testes automatizados muito mais fácil. Finalmente, as ferramentas de refatoração são indispensáveis para automatizar processos que poderiam ser complicados e difíceis se executados manualmente.

Uma das adaptações mais comuns em projetos que adotam XP é relativa à prática do cliente sempre presente, justamente por esta prática exigir uma maior mudança cultural. Nessa adaptação, introduzimos um novo papel, o *Cliente Proxy* na equipe, devido à impossibilidade de se ter um cliente real sempre presente [156, 165, 115]. O Cliente Proxy pode ser um programador ou um gerente que se responsabiliza por fazer a intermediação entre a equipe de desenvolvedores e o cliente real; o papel pode ser alternado por diferentes pessoas. O importante é que o Cliente Proxy esteja presente fisicamente e atue como deve atuar o cliente em condições ideais. Ele responde a perguntas dos desenvolvedores, especifica melhor histórias e toma decisões de pequena escala. Esta prática implica que o Cliente Proxy deve tentar manter contato próximo com o cliente real, reunindo-se presencialmente com ele sempre que possível, mantendo conversas por voz ou, em último caso, utilizando correio eletrônico. O Cliente Proxy deve sempre validar suas decisões com o cliente real e, caso o cliente real tome uma decisão diferente da que foi tomada pelo Cliente Proxy, este deve imediatamente comunicar à equipe que mudou de idéia. A prática é eficaz ao permitir a comunicação com o cliente e disponibilizar um intermediário efetivo para que a equipe possa trabalhar de acordo com aquilo que a metodologia propõe inicialmente.

A prática de retrospectivas [137, 128, 123], ou “oficinas de reflexão” como define Cockburn [29] é anterior às metodologias ágeis e pode agregar muito valor ao processo de implantação de XP em uma organização. A prática consiste em uma atividade que pode ocorrer semanalmente ou mensalmente, após o término de uma iteração ou um *release*, na qual todos envolvidos no processo param para pensar no próprio processo. Durante esta atividade, toda a equipe deve listar de algum modo os acontecimentos bons relativos à maneira como o trabalho foi efetuado e que não queremos esquecer e também aquilo que aconteceu e não foi muito bom (evitamos falar em acontecimentos ruins) e queremos melhorar. É importante ressaltar duas regras desta atividade que existem para minimizar conflitos: não culpar e não citar nomes. Após algum tempo durante o qual todos podem anotar suas impressões, o treinador conduz o processo e lista todos os acontecimentos bons, agrupando eventuais duplicatas. Esta lista é então disponibilizada no espaço informativo da equipe. Acontecimentos

que não foram tão bons também são listadas e agrupados, porém após esta listagem, a equipe deve de alguma maneira determinar quais são os três itens nesta lista que devem ser corrigidos com maior prioridade (isso pode acontecer em uma votação, ou cada membro da equipe pode distribuir um número fixo de pontos entre os itens da lista). Ao determinar os três itens mais prioritários, o treinador então conduz a equipe a listar, para cada um deles, ações concretas que podem ser tomadas para evitar que os problemas recorram. Esta lista de ações concretas deve ser um acordo da equipe e deverá ser aplicada durante a próxima iteração. Esta prática é importante para refletir sobre a adoção da metodologia, descobrir pontos que podem ser melhorados e chegar a um acordo de maneiras como esta melhoria deve ser conduzida [39]. As retrospectivas são importantes para evidenciar para a equipe tudo de bom que aconteceu e também para gerar uma oportunidade na qual insatisfações poderão ser discutidas e erros corrigidos com a ajuda de todos.

## 2.7 A nova versão de XP

No início, a dificuldade de implantar XP em uma organização era justamente o alto acoplamento entre as práticas e o fato de que elas não poderiam ser aplicadas individualmente de maneira segura. Grande parte da resistência à metodologia provinha deste aspecto. Além disso, a comunidade muitas vezes dizia que críticas de quem não fazia XP 100% não eram válidas, pois os defeitos apontados eram devidos a não adesão *completa* à XP.

Após cinco anos de experiência e *feedback*, Kent Beck reescreveu a metodologia de maneira mais positiva e inclusiva, mudando seu discurso [19]. Enquanto no primeiro livro Beck parecia tentar convencer as pessoas a programar de sua maneira, no segundo livro ele tenta explicar as razões que tornam sua maneira de programar mais eficiente e deixa a escolha de adotar ou não cada uma das práticas aos indivíduos. Além disso, ele discorre sobre como é este processo de adoção e provê maneiras mais fáceis de implantar a metodologia incrementalmente.

A evidência desta mudança no teor do discurso é o novo valor, respeito, agregado aos outros quatro: comunicação, simplicidade, coragem e *feedback*. Valorizar o respeito significa valorizar a relação entre membros da equipe e, também, a relação de cada membro com o projeto e sua instituição. Beck ainda explicita que a instituição pode ter outros valores e que o importante é alinhar a equipe com todos estes valores.

Nesta versão, mais importância é dada aos princípios, um instrumento importante que inclusive pode ser utilizado na adaptação de práticas ao contexto local e até mesmo na criação de novas

práticas. Os princípios funcionam como guias, específicos ao domínio da programação, para localizar práticas concretas em harmonia com os valores abstratos.

Todas as doze práticas da primeira edição estão contempladas nesta segunda. Porém elas foram refatoradas para tornar mais simples a adoção incremental da metodologia e para explicitar detalhes que acabavam perdidos em meio à complexidade de algumas práticas anteriores. Beck deixa mais claro que as práticas são objetivos e, no dia-a-dia, nossa execução de uma determinada prática pode não ser exatamente como deveria, pois uma prática depende da situação e do contexto no qual está sendo aplicada. As práticas funcionam, foram validadas através do uso em outros projetos e funcionam melhor quando aplicadas em conjunto. Porém nesta versão as práticas foram divididas em dois grupos, as práticas primárias e as práticas corolário. As práticas primárias são independentes, podem ser aplicadas de maneira segura individualmente e provêm melhorias imediatas ao processo de desenvolvimento da equipe. As práticas corolário são mais difíceis de dominar se as primárias não forem aplicadas e são interdependentes. Beck ressalta que a interação entre as diferentes práticas amplia seus efeitos.

A seguir iremos detalhar os princípios que formam a base da metodologia, as práticas primárias e corolário, a nova abordagem sobre os papéis exercidos pelas pessoas em XP e, finalmente, as mudanças e evoluções em relação à primeira abordagem.

### 2.7.1 Princípios

Os princípios são ferramentas intelectuais para traduzir valores abstratos em práticas concretas pelas quais uma equipe pode se responsabilizar e cuja adoção pode ser medida.

**Humanidade** reconhece que são pessoas com necessidades humanas que desenvolvem software. Destas necessidades, algumas podem ser satisfeitas no trabalho, são elas segurança, crescimento, identidade com o grupo, realização, intimidade e privacidade. Este princípio se concretiza na prática de **trabalho energizado**, com tempo para se dedicar a satisfazer suas outras necessidades fora do trabalho, os seres humanos podem voltar com energia para se dedicar ao trabalho. Um dos desafios proposto por este princípio é balancear as necessidades individuais com as da equipe.

**Economia** busca garantir valor para o negócio. Ao economizar pensamos sobre o valor do dinheiro ao longo do tempo e como melhor empregá-lo. É importante receber o mais cedo possível e gastar o mais tarde possível. É importante também refletir sobre o valor de opções que podemos tomar pela

equipe e pelo sistema, percebendo que a habilidade de poder mudar de idéia no futuro deve guiar nossas decisões. Este princípio é evidenciado nas práticas de **design incremental** e **pague-pelo-uso** e também na priorização e estimativa de **histórias**, garantindo que uma equipe XP não invista em flexibilidade especulativa.

**Benefício Mútuo** é o princípio mais importante e difícil de seguir. Um exemplo de como ele se concretiza é a ênfase dada em XP a testes, refatorações e a metáfora no lugar de extensa documentação escrita. A documentação não traz benefício para os programadores no ato de sua criação, só à organização em um possível futuro. Enquanto investir em testes, refatorações e na construção de uma metáfora trazem benefícios imediatos aos programadores, ao sistema, ao cliente e à organização.

**Auto-semelhança** prega que se deve copiar estruturas e processos existentes para resolver problemas em diferentes contextos ou escalas. É o caso do ritmo similar que observa-se nas práticas do **ciclo de estação**, **ciclo semanal** e nas atividades diárias de programação. Observamos que primeiro criamos testes e depois trabalhamos para que eles funcionem. No ciclo mensal, escrevemos testes de aceitação, que no final do ciclo devem todos passar para validarmos um *release*. Nas atividades diárias escrevemos testes de unidade para validar tarefas que devem ser codificadas.

**Melhoria** indica que não devemos esperar a perfeição mas sim fazer o melhor que podemos hoje, para poder fazer o melhor amanhã. A prática de **ciclo de estação** evidencia este princípio dando à equipe a oportunidade de melhorar o plano de um *release*. A prática de **design incremental** também segue o princípio da melhoria.

**Diversidade** lembra que um time com pessoas diferentes apresenta mais habilidades, conhecimentos e oportunidades. A diversidade é causa de conflitos, sendo importante lidar com essa possibilidade valorizando o respeito. O princípio é evidente nas práticas do **time completo**, que junta diversas habilidades em uma equipe coesa. Evidencia-se também nos diferentes ciclos de planejamento, que dão oportunidade às pessoas trabalharem com suas diversas perspectivas. Aquelas que pensam à longo prazo contribuindo com o **ciclo de estação** e as que tem perspectivas de curto prazo contribuindo com o **ciclo semanal**.

**Reflexão** implica em pensar sobre como e por que trabalhamos. Este princípio pode guiar uma equipe a adotar práticas como a retrospectiva e a realizar análises freqüentes do seu processo de adoção da metodologia. Para isso, é preciso tempo para pensar. Socializar com a equipe em contextos de

diversão ou até em refeições é importante. A reflexão é evidenciada nas práticas do **ciclo de estação** e o **ciclo semanal**, nas conversas de pares programando e na prática de **integração contínua**.

**Fluxo** determina que exista um fluxo contínuo de atividades e que o processo deve explicitar este fluxo. Desta maneira permiti-se que as etapas do desenvolvimento aconteçam em paralelo e não seqüencialmente como proposto em metodologias mais tradicionais. As práticas de **releases freqüentes** e **integração contínua** evidenciam isto.

**Oportunidade** nos leva a encarar problemas como oportunidades para mudança. Estar aberto a oportunidades de aprender e melhorar durante todo o processo é importante.

**Redundância** aumenta as nossas chances de sucesso, promovendo várias oportunidades para fazer a coisa certa. A redundância está presente na complementação das práticas. Outro exemplo são os testes de unidade automatizados: escrevemos o código fonte e, de maneira redundante, escrevemos mais código para verificar se o primeiro funciona, diminuindo a nossa probabilidade de errar.

**Falha** assume que pode ser bom falhar, desde que se aprenda com a experiência. Quando uma equipe não sabe para onde ir, arriscar-se a falhar pode ser o caminho mais curto para obter o sucesso. Este princípio é complementar ao valor de coragem e se evidencia na prática de abandonar código e começar de novo quando percebemos que determinado plano não poderá ser realizado.

**A Qualidade** não deve diminuir nunca. Este princípio prega que quanto maior a qualidade, mais fácil será realizar trabalho. Ele complementa o princípio da humanidade ao satisfazer a necessidade de ser orgulhar do trabalho feito e é evidenciado na prática de controle do escopo no planejamento.

**Passos pequenos** garante que iremos fazer sempre o mínimo que podemos na direção correta, pois a execução de tarefas complexas em passos pequenos diminui o risco. A prática de **design incremental**, **integração contínua**, **implantação incremental** e **implantação diária** refletem este princípio.

**Aceitação de Responsabilidade** evidencia que só o próprio indivíduo pode se responsabilizar por suas ações. Este princípio está claro na prática de estimativas feitas pelos próprios programadores no jogo do planejamento.

### 2.7.2 Práticas primárias

As práticas primárias são guias para se começar a adoção de Programação eXtrema em uma organização. Elas podem ser implantadas facilmente pois são seguras e devem ser introduzidas em pequenos passos, para evitar uma mudança muito rápida na cultura da organização. É importante ter tempo para que os novos hábitos sejam incorporados pela equipe. O uso de métricas para aumentar a consciência sobre a necessidade ou benefícios de uma prática é recomendado e a avaliação da experiência, usando retrospectivas por exemplo, também. Das práticas originais, a **programação pareada** e a **integração contínua** se mantêm sem mudanças relevantes. Além destas duas, a prática de **ritmo sustentável** foi renomeada novamente e agora se chama **trabalho energizado**, eliminando críticas à quantidade de horas que representam trabalho produtivo e sustentável. A prática de **testes automatizados** foi rebatizada de **desenvolvimento dirigido por testes** e inclui a prática de **refatoração**. Além disso a prática de **design incremental** engloba as práticas de **metáfora** e novamente **refatoração** em uma atividade diária. A seguir iremos detalhar as novas práticas.

**Sentar Juntos** deixa explícita a necessidade de se ter um espaço onde toda a equipe possa trabalhar junta, valorizando a comunicação e possibilitando que as pessoas possam se beneficiar de todos seus sentidos ao conversar. Ressalta-se a necessidade também de pequenos espaços privativos para respeitar o princípio de humanidade.

**Time Completo** lembra que a equipe precisa de pessoas com todas as habilidades e perspectivas necessárias para o sucesso do projeto. Nesta segunda versão, Beck remove a limitação de tamanho das equipes e ressalta somente dois limites naturais: equipes de 12 pessoas é o limite natural de pessoas com as quais um indivíduo pode interagir em um dia e equipes de 150 pessoas é o limite natural para que um indivíduo se lembre de todos outros no time. Beck recomenda que se fracione problemas complexos e que estes sejam resolvidos com times de times que respeitem estes limites.

**Espaço de trabalho informativo** era uma prática implícita que ganhou destaque na segunda edição. Esta prática diz que qualquer observador interessado deve ser capaz de olhar para o espaço de trabalho e ter idéia do andamento do projeto em pouco tempo. O espaço informativo deve conter cartazes grandes e visíveis, que comunicam métricas coletadas pelo acompanhador. Limpeza, ordem, espaço para programação pareada e disponibilidade de água e comida garantem que o espaço informativo complemente a prática de trabalho energizado.

**Histórias** que antes faziam parte do jogo do planejamento, viram uma prática independente na segunda versão. Elas continuam iguais, textos simples que expressam necessidades do cliente e que são estimadas, o quanto antes, pelos programadores.

**Ciclo semanal** que antes era parte do jogo do planejamento, explicita as iterações que fazem parte de um *release*. O ciclo semanal inclui um jogo de planejamento do trabalho que deve ser efetuado em aproximadamente uma semana, ou seja, planeja as iterações de menor granulosidade. Uma reunião no começo da semana revê o progresso de um *release*, comparando o que foi feito com o que tinha sido planejado na semana anterior. O cliente prioriza uma semana de histórias, que são divididas em tarefas, que por sua vez são aceitas e estimadas pelos programadores. A semana começa com a escrita de testes de aceitação da iteração e acaba com a implantação do sistema codificado. Beck reconhece que este planejamento custa tempo, mas que esse é um gasto necessário.

**Ciclo de estação (*Quarterly Cycle*)** que antes era parte do jogo do planejamento, explicita o planejamento de um *release*. A avaliação de um ciclo de estação pode identificar gargalos e dependências da equipe com outras partes da organização, apresentando uma oportunidade para melhorias e reparos. Durante o planejamento deste ciclo, temas são escolhidos para a estação. Estes temas servem para agrupar histórias relacionadas, que também são criadas neste momento. O uso de temas garante que não há exagero de detalhes e que o planejamento acontece com a perspectiva do longo prazo. O enfoque do planejamento da estação é também perceber como o projeto se encaixa com o resto da organização. A estação pode ter duração variável e depende do contexto do negócio. Um trimestre é uma boa medida usada para medir progresso em muitas organizações.

**Folga** reconhece que o planejamento, por melhor que tenha sido, sempre falha. Para evitar atrasos ou a necessidade de renegociação de escopo, o planejamento deve conter explicitamente espaços de folga. Isto pode acontecer tanto com a introdução de tarefas menores, que podem ser descartadas em caso de atraso, quanto com a distribuição de histórias de tempo livre para os programadores. Se o progresso de um ciclo for bom, este tempo pode ser usado para pesquisa e para manter o trabalho energizado. Caso atrasos ocorram, a folga pode ser descartada e mais trabalho pode ser realizado para que a equipe consiga entregar as histórias que prometeu ao cliente.

**Build veloz** exige que o sistema deve ser compilado por completo, de maneira automática e todos os testes devem ser executados, em no máximo 10 minutos. Esta prática provê agilidade à equipe e complementa a habilidade de entregar *releases pequenos*. [O limite de 10 minutos é somente o tempo](#)

razoável para que o par possa tomar um café durante o *build*, porém não é essencial. Se o build demora menos que 10 minutos, excelente, o café pode ficar para depois. Se demora mais existem duas possibilidades. A primeira é de que o *build* pode ser refatorado, para que não rode todos os testes de aceitação por exemplo, pois estes podem demorar muito tempo para executar. A segunda é de que o *build* é lento pois o sistema é muito complexo, e a demora pode ser um indício para simplificá-lo.

**Design incremental** A prática de *design* simples rendeu críticas a XP que diziam que a metodologia não investia em *design* da aplicação. Para responder às críticas, Beck a redefine como *design* incremental. Esta prática implica em investimento diário no *design* da aplicação e justifica o fluxo contínuo de *design* ao mostrar que, se realizado mais perto de quando será utilizado, ele tende a ser mais eficiente e valioso.

### 2.7.3 Práticas corolário

As práticas corolário são difíceis ou perigosas de serem implementadas sem antes serem dominadas as práticas primárias. O conselho de progredir em direção às práticas incrementalmente se mantém para organizações adotando XP. A prática do **cliente sempre presente** foi renomeada para **envolvimento real com o cliente** e assume que muitas vezes um cliente direto não poderá ser colocado com a equipe. A prática de **propriedade coletiva do código** foi renomeada para **código compartilhado** e se mantém como prática corolário. A seguir iremos detalhar as novas práticas.

**Implantação incremental** lida com uma equipe que deve substituir um sistema legado e diz que essa substituição deve ser feita incrementalmente. O importante é manter o sistema funcionando e ter segurança na migração. É possível que, durante algum tempo, ambos o legado e o novo sistema devam funcionar em conjunto, adicionando um pouco de trabalho de comunicação extra tanto no sistema quanto com usuários, mas garantindo harmonia na migração.

**Continuidade da equipe** propõe que equipes eficientes continuem trabalhando juntas. Deve-se incentivar um rodízio razoável entre equipes, mas ao se concentrar na eficiência da organização como um todo, o valor de equipes que trabalhem bem juntas se torna evidente.

**Redução da equipe** Com o passar do tempo, a necessidade de uma equipe grande pode diminuir, principalmente se o sistema entra em um ciclo de manutenção. Se isto acontecer, mantenha a carga de trabalho constante e distribua tarefas de modo a deixar alguém ocioso, esta pessoa pode ser

liberada para formar novos times. Esta prática tende a eliminar o desperdício e ajudar a organização a resolver novos problemas.

**Análise da causa inicial** Sempre que encontrar um defeito, elimine o defeito e sua causa, para que o mesmo tipo de erro não ocorra novamente. Esta prática define uma nova atividade, que deve acontecer quando surgir um defeito. Ao encontrar um defeito, um par deve primeiro escrever um teste de aceitação automatizado que evidencie o erro no nível do sistema e então escrever um teste de unidade no menor escopo possível. O par deve proceder para resolver o problema e passar nos testes. O último passo é descobrir por que o defeito surgiu e, principalmente, como ele passou despercebido. A técnica de análise da causa inicial propõe que se pergunte 5 vezes o porquê do defeito ter surgido e sugere que a causa inicial, na maioria das vezes, é um problema relacionado à equipe e que pode ser resolvido utilizando-se práticas que evitem que ele recorra.

**Código e testes** explicita que só o código fonte e os testes automatizados devem ser artefatos permanentes gerados por uma equipe XP. Até mesmo as histórias e cartazes devem ser descartados, pois o histórico do projeto se mantém por mecanismos sociais. A cerimônia envolvida em documentação interrompe o fluxo de valor.

**Repositório único de código** complementa a prática de código compartilhado e vai além, dizendo que todo o código deve estar contido em um único repositório que não deve ter *branches* permanentes. O problema com múltiplos repositórios é que eles não escalam. Se o seu sistema é tão complexo que precisa de repositórios separados, isso é evidência de um problema no *design*.

**Implantação diária** é a evolução da prática de *releases* pequenos e é ainda mais extrema. Ela determina que o seu sistema deve ser implantado diariamente, de preferência com auxílio do *build* veloz. O objetivo é colocar histórias implementadas em produção toda noite, para que os usuários possam usufruir de benefícios o quanto antes. Esta prática depende de uma baixa taxa de defeitos e de um processo automático de implantação, com habilidade para implantação incremental e eventuais *rollbacks* em caso de falhas.

**Contrato de escopo negociável** determina como devem ser feitos contratos em um projeto que adota XP, fixando o tempo, custos e a qualidade, mas mantendo o escopo negociável. Desta maneira, o escopo é negociado constantemente com o cliente, possivelmente no planejamento do ciclo de estação e a equipe pode celebrar uma seqüência de contratos curtos com o cliente. Vinicius Teles

disponibilizou [154] um modelo de contrato de escopo negociável que poder ser usado na legislação vigente no Brasil.

**Pagar pelo uso** também aborda o lado de negócios de um projeto XP, desta vez sugerindo que o cliente pague por toda vez que for usar o sistema, trazendo à tona o debate sobre “arquiteturas orientadas a serviços” que está em voga na indústria atualmente. Esta prática valoriza o dinheiro como *feedback* mais importante e provê ao cliente possibilidades de prever custos.

#### 2.7.4 Papéis

Nesta nova versão, Beck inclui todos as funções que tipicamente se encontram em uma organização que desenvolve software e explicita quais são os papéis que cada uma pode assumir para colaborar com uma equipe XP. A diversidade de papéis traz benefício à prática de time completo. Cada parte no grupo deve entender seu papel no todo e a interação entre as pessoas deve seguir os princípios de fluxo e benefício mútuo. Uma pessoa pode assumir mais de um papel e os papéis podem ser revezados entre pessoas.

Colaborando no papel de clientes *proxy* e com planejamento, o nível mais alto da organização conta com **Gerentes de Produto** que escrevem histórias, escolhem temas para o planejamento de estação e ajudam a priorizar histórias no jogo do planejamento, representando as necessidades dos clientes reais. Estes são auxiliados por **Gerentes de Projeto** que facilitam a comunicação dentro do time, coordenam a comunicação com o cliente, marcam progresso e mantém os planos da equipe sincronizados com a realidade da organização. **Gerentes de Recursos Humanos** trabalham em contratações e revisões, dando ênfase ao trabalho em grupo e habilidades sociais de novos contratados para garantir que a equipe funcione bem junta. Os gerentes devem atuar conectando pessoas e não como gargalos. Finalmente, **Executivos** articulam e mantém objetivos de larga escala, monitoram o andamento do projeto, incentivam e facilitam melhorias dentro da equipe, são essenciais para criar um ambiente de apoio à equipe XP dentro da organização.

Os **Usuários**, se acessíveis, são importantes para ajudar a escrever e escolher histórias e tomar decisões no papel de clientes *proxy* durante o desenvolvimento, atentos ao fato de que representam uma comunidade maior. Eles se relacionam com **Projetistas de interface**, que devem ajudar na escolha da metáfora do sistema, ajudar o cliente ou cliente *proxy* a escrever histórias e avaliar o sistema entregue para priorizar os próximos passos. Ambos se relacionam ainda com **Escritores Técnicos** que provêm *feedback* de funcionalidades implementadas, criando uma relação próxima

com os reais usuários de um sistema. Estes últimos escrevem manuais e tutoriais e verificam se estes documentos estão sendo utilizados, também ajudando a escolher histórias no planejamento.

Programadores guiam o processo de desenvolvimento, estimando histórias e tarefas, escrevendo testes e código, executando o *build* automático, melhorando o design gradualmente e praticando programação pareada. Eles são auxiliados por **Arquitetos de Software** que procuram oportunidades de executar refatorações em larga escala, escrevem testes para estressar a arquitetura, implementam histórias investindo em *design* incremental, direcionam a evolução da arquitetura e particionam o sistema para simplificá-lo. Finalmente, **Testadores** ajudam o cliente a escolher e escrever testes de aceitação, também ensinando outros programadores técnicas de testes e sempre concentrando-se no que deve acontecer quando der errado.

É importante notar que os papéis não são fixos e que cada um deve contribuir com tudo que pode para a equipe.

### 2.7.5 Mudanças e evoluções

Alguns outros aspectos de XP também mudaram nesta nova versão. Em particular, o planejamento agora sugere estimativas em tempo real, facilitando o processo que antes usava pontos. Quanto aos testes, agora não se exige que sejam realizados antes de codificar, desde que sejam realizados o quanto antes.

Uma das grandes mudanças foi assumir que XP escala para times maiores, quando a primeira versão restringia o número de desenvolvedores, aplicar métodos ágeis em grande escala foi um tópico de pesquisa explorado [34]. XP também funciona para equipes distribuídas, com as devidas adaptações das práticas.

Além disso, nesta nova versão a implantação da metodologia ficou mais clara como um processo incremental, adaptado ao contexto local.

A Figura 2.3 mostra como as práticas evoluíram da primeira para a segunda versão de XP:

A cor da fonte indica se a prática é da primeira versão (em preto), e na segunda versão se é primária (em azul) ou corolário (em verde). Podemos observar que as práticas que foram mantidas entre versões se tornaram todas práticas primárias (em degrade do preto ao azul). Em grupos de cores, podemos ver como as práticas da primeira versão foram repensadas e estão compensadas nas práticas da segunda versão.



Figura 2.3: A evolução das práticas de XP



## Capítulo 3

# Experiências com o ensino de XP

Como já vimos, existem muitos relatos de aplicações bem sucedidas de XP [119, 90, 42, 135, 60, 24, 71, 86, 11, 75]. Estes casos porém não se aprofundam nas questões relevantes ao nosso trabalho, refletir sobre o ensino de XP e oferecer sugestões de como melhorá-lo. Também vimos que o ensino de algumas práticas, em particular aquelas consideradas primárias na segunda versão, como testes, refatoração, programação pareada e integração contínua, já foi bem explorado. Porém, a sugestão mais freqüente é de como ensinar estas práticas de maneira isolada.

Vimos que o valor da Programação eXtrema está justamente na sinergia entre as diversas práticas e defendemos que a melhor maneira de ensiná-la é apresentando o todo [14] de maneira prática [77]. Mesmo assim, existem diversas abordagens que podem ser utilizadas ao encarar o desafio de ensinar XP: palestras, jogos, cursos teóricos, *workshops* rápidos de um ou dois dias, cursos de média duração e cursos mais longos.

Palestras, jogos e cursos teóricos são uma boa opção para divulgar informação sobre a metodologia, além de serem úteis e eficientes no ensino de práticas isoladas. Cursos médios normalmente têm duração de poucas semanas e dão enfoque a atividades práticas. Esses, em conjunto com palestras, jogos, cursos teóricos e *workshops*, podem ser oferecidos em uma variedade de contextos. Já cursos mais longos deverão ser aplicados levando em conta o contexto no qual irão acontecer, na academia podendo fazer parte do currículo de ensino de computação, com duração mínima de um semestre e desenvolvimento de aplicações reais e em outros contextos como na indústria ou governo, podendo acontecer como consultorias e *mentoring*.

Estas diferentes modalidades de ensino nos levam a refletir sobre a diferença entre ensinar e aplicar um método e ensinar uma metodologia. Obviamente, o objetivo final do educador deve influenciá-lo

na escolha das diversas opções. Palestras, jogos e cursos mais curtos bastam se o objetivo é ensinar XP como método, apresentando uma receita relativamente fechada que poderá ser aplicada por completo, ou não, de acordo com a necessidade e vontade dos alunos após a experiência. Já se pretendemos qualificar um grupo ou organização com ferramentas e técnicas para que possam evoluir e adaptar o seu uso do método de maneira autônoma, devemos ter ciência de que estamos de fato ensinando XP como metodologia. Nesse caso, somente experiências mais longas e práticas serão eficazes.

Neste capítulo, apresentaremos uma síntese das experiências de ensino de XP em diferentes contextos, começando com evidências da literatura e seguindo com nossos estudos de caso. No próximo capítulo, refletiremos sobre essas experiências.

### 3.1 Trabalhos relacionados na Academia

Atualmente, observamos que a maioria dos cursos de graduação em Ciência da Computação no Brasil e no exterior são baseados em métodos convencionais que tem enfoque em ferramentas, documentação, contratos e a elaboração e cumprimento de planos. Engenharia de Software é matéria obrigatória do currículo, têm o objetivo de preparar alunos para o mercado de trabalho e projetos de maior escala e complexidade. Nessa tentativa, a disciplina normalmente adota o modelo em cascata e o aplica em projetos de pequenos grupos, nos quais o maior enfoque é familiarizar alunos com a documentação que precisa ser gerada e as etapas de processos tradicionais, não criar software complexo e de qualidade.

Em 2004, uma discussão no simpósio técnico sobre ensino de Ciência da Computação da ACM, intitulada *“Teaching software development methods: The case of Extreme Programming”*, trouxe à tona o tópico de gradualmente introduzir algumas das práticas de XP no currículo da disciplina [66]. Acreditamos que esta iniciativa é válida e importante, tendo em vista a quantidade de experiências que relatam o ensino da metodologia na academia.

Em uma das primeiras experiências acadêmicas com XP relatadas, Wilson [163] apresenta um projeto onde 20 alunos do quarto ano melhoraram um protótipo de IDE Java enquanto ele atuava como treinador e cliente. O curso durou um semestre e contava com sessões semanais de programação com duração de 3 horas e um Wiki para suporte. Programação pareada foi adotada, mas testes e refatorações podiam ser efetuados individualmente. A equipe não contava com um laboratório dedicado. As práticas de metáfora e cliente sempre presente não foram adotadas. A prática de testes teve resistência dos alunos no início, mas foi bem incorporada ao final do curso.

Além deste caso, algumas experiências utilizando poucas práticas de XP tiveram sucesso relativo. Astrachan et al. [13] relatam uma experiência de curso universitário onde poucas práticas foram adotadas no desenvolvimento de projetos triviais durando 3 semanas e meia com 180 alunos do quarto ano. Programação pareada era usada entre o instrutor e todos os alunos, ao mesmo tempo. Refatoração e *releases* pequenos foram as outras práticas adotadas. Lappo [85] ensinou XP a um grupo de estudantes de mestrado, atuando como treinador e cliente. Produziram uma aplicação Web de gestão de recursos usando Java, em regime de dedicação integral durante 12 semanas nas quais trabalhavam em duas sessões de 3 horas. Segundo sua avaliação, somente as práticas de jogo do planejamento e ritmo sustentável foram de fato eficientes. As práticas de integração contínua e propriedade coletiva do código não foram adotadas por resistência dos alunos. Conclui seu estudo apontando que alunos que não passaram por experiências com métodos tradicionais não dão valor às diferenças propostas por métodos ágeis. Noll e Atkinson [117] descrevem uma experiência em um curso de 10 semanas no qual equipes seguindo metodologias tradicionais conseguiram entregar mais funcionalidades (e também mais defeitos) do que equipes que optaram por XP, que porém produziram código mais robusto. Nota-se que as equipes “ágeis” não conseguiram seguir algumas práticas como envolvimento real com o cliente, integração contínua e propriedade coletiva do código. Apóiam a hipótese de Lappo que alunos dão valor aos métodos ágeis somente após sofrer com métodos tradicionais.

Com uma visão mais crítica sobre o valor de XP, Schneider e Johnston [133, 134] utilizaram o método parcialmente em projetos finais de 15 alunos de quarto ano, com duração de 2 semestres. Os alunos trabalhavam em duas equipes, juntos por 10 horas semanais em um laboratório dedicado. Os autores apontam que nem todas práticas de XP são adaptáveis ao contexto acadêmico, algumas indo contra objetivos educacionais e por isso não recomendam XP como método a ser ensinado na academia, em especial pela percepção de alunos de que XP não investe em análise de requisitos e *design*.

Shukla e Williams [139] ensinaram XP em um curso que apresentava diversos métodos a 150 alunos que trabalhavam em um jogo. Durante 4 semanas os alunos adotaram XP como método, porém as práticas de metáfora e refatoração não foram utilizadas. Relatam dificuldades com a adoção de testes pelos alunos, apontando que alguns grupos só codificaram testes automatizados no final do projeto para obter nota. Em especial, testes de aceitação tiveram muita resistência, mesmo sendo escritos pelos próprios professores e utilizados para avaliar o desempenho dos projetos. Concluem que XP não pode ser apresentada sem complementos, em específico técnicas de UML e sugerem um método

híbrido no qual algumas práticas tradicionais seriam utilizadas em conjunto com práticas ágeis para não renegar aspectos considerados importantes no *design* de aplicações.

Já Noble et al. [116] relatam uma experiência na Nova Zelândia na qual estudantes cursando a disciplina de projeto de conclusão (historicamente centrada em documentos na qual estes valiam 80% da nota enquanto software funcional contava somente 10%) em grupos de 4 a 7 pessoas poderiam optar por utilizar Programação eXtrema nas doze semanas em que desenvolveriam um projeto para clientes externos com alunos do ano anterior atuando como treinadores. Ressaltam que os grupos que optaram pela metodologia ágil produziram software de mais qualidade e trabalharam menos tempo em comparação aos outros.

Com uma visão mais otimista, Holcombe et al [70] relatam adaptações a um curso existente, no qual 25 alunos do quarto ano em grupos de 5 alunos participam de uma empresa junior durante 10 horas semanais, com clientes reais de empresas parceiras da universidade, em 12 semanas de trabalho. Ele aponta que clientes reais são essenciais para o processo. Continuando este estudo Syed-Abdullah et al. [144] apontam que o sucesso de projetos é diretamente proporcional ao número de práticas de XP adotadas, em especial a prática de pequenos *releases* aumenta a satisfação dos clientes com o processo.

Apontando XP como uma alternativa viável, Tomek [151] apresenta sua experiência de ensino de XP em dois pequenos cursos de Ciência da Computação para grupos de 4 e 8 alunos trabalhando 3 horas semanais durante um semestre. Ele ressalta a importância de criar um ambiente ágil e para isso usa o VisualWorks Smalltalk. No seu primeiro curso, dois projetos foram desenvolvidos, um inicial para que os alunos se ambientassem com XP, seguido de um projeto realístico. Em ambos atuou como treinador e cliente. No segundo curso, concentrou-se em um único projeto com um cliente verdadeiro. Ele recomenda que algumas práticas de XP sejam introduzidas mais cedo na grade curricular universitária e que as disciplinas de Engenharia de Software e de projeto de conclusão podem ser mais efetivas adotando XP, ressaltando a importância de adotar todas as práticas e de contar com um cliente externo.

Pechau et al. [20] relatam a experiência de oferecer um projeto de três semanas aos alunos no final do primeiro ano de Ciência da Computação em uma universidade na Alemanha. Neste projeto, uma semana de programação intensa se baseia nas práticas de XP, na qual 3 grupos de 12 a 18 alunos treinados por professores com experiência em XP, que também atuam como clientes, tem sua própria sala de trabalho. Nota-se que alunos de primeiro ano ainda não têm a experiência necessária

para compreender todas as práticas, porém aprendem muito sobre trabalhar em grupo e sentem-se motivados pelo projeto. McKinney et al. relatam experiência semelhante com alunos de primeiro ano [101], porém em um curso de duração de um semestre e notam que o instrutor não deveria se sobrecarregar com o papel de cliente.

Mugridge et al. [109] descrevem experiências mais bem sucedidas em 3 instâncias de um curso anual, porém notam dificuldades com as práticas de ritmo sustentável e envolvimento real com o cliente. Muller et al. [111] ressaltam a importância do treinador no contexto universitário para ajudar na prática de testes e de implantação incremental e ampliam seus estudos para propor uma semana dedicada ao desenvolvimento de um projeto prático ao invés de pequenas aulas práticas ao longo do curso [110].

Hedin et al. [68] também ressaltam a importância de treinadores dedicados em um curso de 7 semanas com 107 alunos de segundo ano que se encontravam durante 8 horas em um dia por semana, cujo projeto começou com uma base de código oferecida pelos professores. Eles apontam o sucesso de usar alunos da instância da disciplina no ano anterior como treinadores. Observam os mesmos problemas da maioria das experiências aqui citadas, como dificuldades para ensinar testes, adotar corretamente a prática de *design* simples e sobrecarga de trabalho do professor que assume papel de cliente. Ressaltam ainda a importância de alocar as equipes em salas dedicadas ao projeto. Aprimorando seus estudos, Hedin et al. relatam sua experiência positiva ensinando grandes grupos de alunos em duas disciplinas que funcionam em conjunto, uma sobre programação em grupo utilizando XP (para 100 alunos) e outra específico para 25 alunos (que cursaram a primeira) na qual assumem o papel de treinadores da primeira disciplina [67, 69].

Em sua pesquisa, Dubinsky [65, 44, 45] relata experiências em 5 cursos semestrais, nos quais 325 alunos trabalharam em 25 projetos, e sugere alguns princípios de ensino de métodos de desenvolvimento e como eles são evidenciados no ensino de XP. Dubinsky recomenda ainda que cada membro das equipes, além de atuar como desenvolvedor, tenha um papel especial relacionado às práticas de XP (por exemplo: responsável por testes de aceitação) para que alunos se responsabilizem ainda mais pelo processo.

Finalmente, Muller [113] relata dificuldades encontradas ao adotar XP em um projeto complexo, durante 6 meses com uma pequena equipe de 4 alunos, que desenvolveu uma máquina virtual Java. Devido à alta complexidade algorítmica do projeto, algumas adaptações ao jogo do planejamento foram propostas.

Na próxima sessão descreveremos em detalhes a experiência do nosso grupo ensinando XP em um contexto acadêmico nos últimos 7 anos [63].

### 3.2 Laboratório de XP - 2001 a 2007

Desde 2001 a disciplina **Laboratório de Programação eXtrema** é oferecida como optativa no currículo de graduação e pós-graduação em Ciência da Computação do IME/USP [63, 54, 53]. A disciplina é voltada à prática, dura um semestre, conta com poucas aulas teóricas que descrevem o método no início do curso e de duas a quatro sessões práticas por semana, durante quatro meses. O curso é voltado especialmente para alunos do terceiro e quarto ano da graduação, já com alguma experiência de desenvolvimento e alunos de pós-graduação. Ao longo das instâncias do curso o número de participantes inscritos variou entre 6 e 42 pessoas, que na média se organizaram em grupos de 6, fato que em si é relevante por oferecer aos alunos a oportunidade de trabalhar em grupos grandes, já que a maioria das disciplinas trabalha com grupos muito menores. Em sete anos, a disciplina foi oferecida seis vezes, e 18 equipes trabalharam em 9 projetos distintos. No total 120 alunos foram beneficiados pela disciplina.

Em sua primeira instância, no segundo semestre de 2001, dois grupos de seis alunos desenvolveram soluções distintas para um mesmo sistema de controle de carga didática sugerido pelos professores. O sistema deveria contar com uma interface para coletar informações sobre as disciplinas oferecidas por um departamento de uma universidade, além das necessidades e desejos de alunos e professores. Com estas informações, o sistema deveria sugerir um possível horário para as disciplinas com seus respectivos instrutores. Apenas um dos grupos conseguiu colocar parte do sistema em produção [54]. Iremos detalhar esta primeira experiência, na qual participei como aluno, mais adiante.

Em 2002, os professores sugeriram que um grupo de seis alunos desenvolvessem um sistema para resolver o problema de escolha dos livros a serem comprados pelo departamento de Computação, a partir de critérios escolhidos pelo operador do sistema.

Em 2003, os alunos, divididos em equipes de 8 e 9 pessoas, desenvolveram módulos diferentes de um sistema de controle do acervo da biblioteca do IME/USP, o Colméia. O primeiro grupo desenvolveu o controle dos periódicos da biblioteca, enquanto o segundo o módulo de empréstimo de livros e outros materiais. Esta foi a primeira experiência onde um cliente externo, o administrador da biblioteca do Instituto, participou do laboratório.

Em 2004, 3 equipes, de 3, 8 e 11 alunos participaram da disciplina. A primeira equipe desenvolveu

um sistema em Smalltalk para marcar reuniões. A segunda continuou o desenvolvimento do Colméia, iniciado no ano anterior. A terceira desenvolveu um software livre para distribuição e compartilhamento de arquivos multimídia, para ser utilizado no projeto “Pontos de Cultura” do Ministério da Cultura, foi a primeira experiência com clientes externos ao IME [53]. Estes projetos, nos quais atuei como auxiliar de ensino, serão descritos em mais detalhes a seguir.

Em 2005 a disciplina não foi oferecida devido a uma mudança da grade de horários, a partir deste ano ela passou a ser oferecida no primeiro semestre, evitando concorrência com outras disciplinas optativas disputadas pelos alunos.

Em 2006 a disciplina contou com 5 equipes. Uma equipe com 8 alunos continuou o desenvolvimento do Colméia. Outra equipe de 8 alunos se envolveu com o desenvolvimento de um software CAD de código aberto para desenho técnico e de arquitetura, o Archimedes, cuja base de código era software livre e tinha sido desenvolvida usando XP. Uma equipe de 4 alunos deu continuidade ao Borboleta, um sistema iniciado como projeto de formatura em 2004, uma plataforma móvel para auxílio a médicos que fazem atendimento domiciliar. Uma equipe de 6 alunos iniciou o desenvolvimento de um aplicativo para auxiliar no tratamento e prevenção de Lesões por Esforço Repetitivo (LER), o GinLab. Finalmente, uma equipe de 6 alunos desenvolveu um sistema que deverá fazer parte de um *grid* computacional e realizar conversões entre diferentes formatos e codificações de vídeo.

Em 2007, 43 alunos participaram da disciplina. Nesta instância todos projetos foram continuação de projetos anteriores do laboratório. O desenvolvimento do Archimedes continuou com um grupo de 9 pessoas. O Borboleta foi aprimorado por um grupo de 11. O GinLab prosseguiu com 9 alunos. O Colméia contou com dois grupos, um de 10 pessoas que desenvolveu o módulo de buscas, e outro de 4 pessoas que trabalharam em um módulo específico, o Zumbido, responsável por transferir dados entre bibliotecas diferentes.

A seguir iremos descrever o funcionamento do laboratório, aspectos acadêmicos e adaptações das práticas. Após esta exposição iremos descrever em mais detalhes um dos projetos do primeiro ano, no qual fui aluno, e os três projetos do quarto ano, quando atuei como auxiliar de ensino.

### 3.2.1 Como funciona o Laboratório

A disciplina conta com um laboratório especialmente preparado e de uso exclusivo durante as aulas, na Figura 3.1 vemos o espaço disponibilizado para uma equipe e na Figura 3.2 vemos outra equipe ocupando suas estações de programação pareada.



Figura 3.1: O espaço do laboratório.

O laboratório conta com sua própria rede de computadores configurados com sistema Debian GNU/Linux com ambientes de desenvolvimento baseados em Eclipse devidamente instalados. Além disso, um servidor possibilita acesso externo à rede, fornece serviços de integração contínua como CVS e Subversion e disponibiliza sistemas para organização das equipes como Wikis e o XPlanner. Vale ressaltar que o laboratório conta com alunos bolsistas que cuidam da administração e manutenção desta infra-estrutura.

Este laboratório foi montado seguindo sugestões de Cockburn para aumentar a eficiência da comunicação (ver o Capítulo 3 *Communicating, Cooperating Teams* [29]). Há espaço para dois desenvolvedores sentarem em cada computador, e membros de uma mesma equipe sentam de frente uns para os outros. Na Figura 3.3 vemos dois pares atuando.



Figura 3.2: Uma equipe ocupa suas estações de programação pareada no laboratório.

Além disso, cada equipe conta com espaço nas paredes para posicionar radiadores de informação e ainda espaço em um quadro branco, que também é usado para separar as diversas equipes. Desta maneira o espaço do laboratório permite comunicação entre membros de diversas equipes (durante o almoço, conversas e através dos radiadores de informação) e comunicação mais eficiente entre membros de uma equipe, devido ao posicionamento das mesas, do quadro e os computadores. Nas figuras 3.4 e 3.5 vemos os radiadores de informação.

A carga didática da disciplina reflete o contexto acadêmico, os alunos devem comparecer a duas aulas semanais uma de 2 horas e outra de 3 horas, sessões práticas nas quais desenvolvem seu projeto de acordo com XP, adotando todas as práticas. Diferente da maioria das disciplinas, optamos por não realizar provas. A nota dos alunos é composta principalmente pelo engajamento em seguir o método (35%), pela frequência nas aulas (30%), a qualidade do software produzido (25%) e uma



Figura 3.3: Outro par desenvolvendo seu sistema.

auto-avaliação (10%). Fica claro que o maior objetivo do curso é que alunos se dediquem a aplicar a metodologia e compareçam a todas as aulas.

Normalmente, a aula de 3 horas acontece logo antes do almoço, que oferecemos aos alunos para que possam aproveitar e alongar sessões de programação pareada, denominado “almoço extremo”. Observamos que estas sessões são mais produtivas do que as de 2 horas, e que o almoço extremo, porém modesto, ajuda os alunos a relaxarem criando uma oportunidade para aumentar a comunicação entre eles, que conversam sobre os sistemas e trabalham em pares com lanches em mãos. A importância de comida no ambiente de trabalho foi ressaltada por Beck [16]. Na Figura 3.6 vemos dois professores que também atuam como clientes discutindo os projetos durante o almoço extremo.

Além das duas aulas mencionadas, é fortemente sugerido aos alunos que trabalhem de duas a

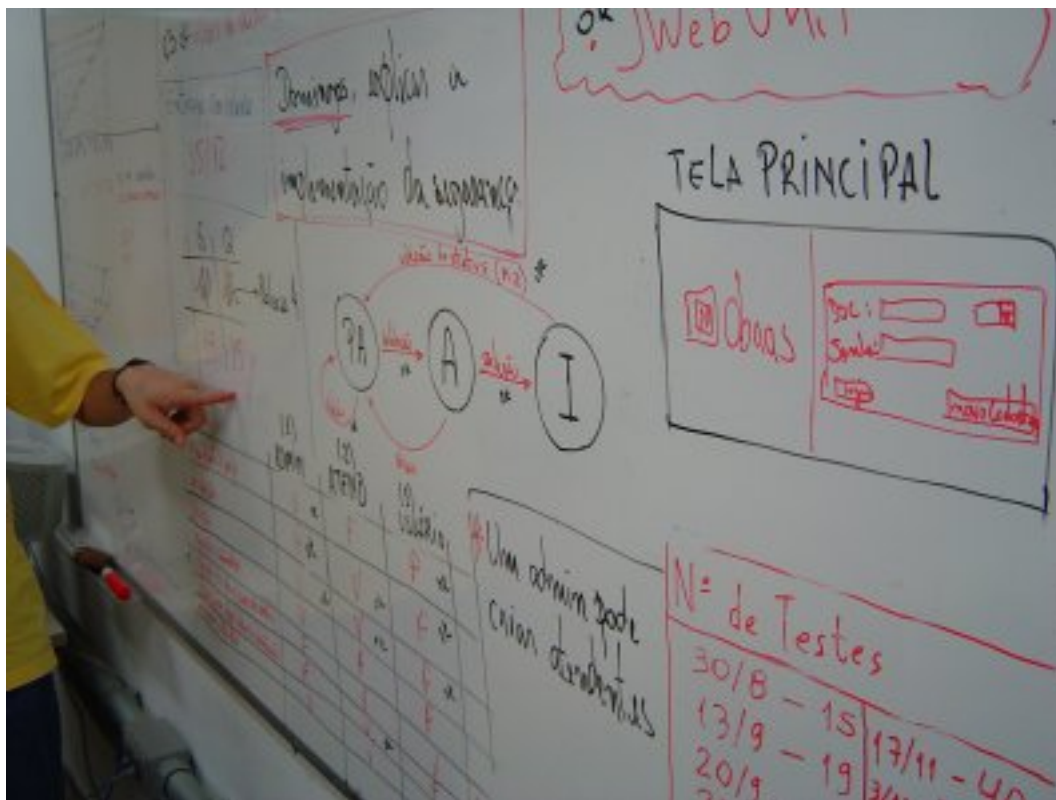


Figura 3.4: O quadro branco de uma equipe sendo utilizado como radiador de informações.

quatro horas extras por semana no laboratório, em sessões de programação pareada ou pesquisando tecnologias usadas nos projetos. Porém, estas horas não são contabilizadas pelos professores, e não valem presença para avaliação do curso. Na prática, os alunos trabalham no laboratório, também trabalham em casa e chegam até a fazer sessões de programação pareada remotamente. Porém, alguns alunos não se dedicam as horas extras justamente por não serem obrigatórias.

Para formar equipes, os professores sugerem sistemas que poderão ser desenvolvidos e alunos se agrupam de acordo com seus interesses. A escolha final fica a critério dos alunos, que também podem sugerir outros projetos e tem autonomia sobre a escolha das tecnologias que serão usadas no desenvolvimento.

Foi importante escolher bem os sistemas que iríamos sugerir, primeiro para que existissem de



Figura 3.5: Radiadores de informações colados na parede.

fato clientes e potenciais usuários para interagir com a equipe e segundo para que fossem desafios interessantes do ponto de vista tecnológico e motivassem os alunos.

Em cada equipe um aluno escolhe atuar como acompanhador, voluntariamente, e fica responsável por atualizar os radiadores de informação de seu grupo. Os acompanhadores recebem literatura específica sobre seu papel e são sempre motivados a desempenhar suas funções. Os acompanhadores também mantêm, normalmente, versões *on-line* das histórias relativas aos seus sistemas. Algumas vezes surge uma competição saudável entre os acompanhadores de diversas equipes transformando o laboratório em uma verdadeira exposição de radiadores de informação. A Figura 3.9 mostra o acompanhamento da evolução do número de testes de um projeto e a Figura 3.7 mostra histórias documentadas *on-line* no XPlanner visualizadas no sistema que roda no laboratório.



Figura 3.6: Dois professores que também atuam como clientes refletem sobre os projetos em andamento durante o almoço.

Nas primeiras instâncias da disciplina, professores atuavam como treinadores e também como clientes, já que seriam usuários dos sistemas desenvolvidos. Ao longo dos anos evoluímos nossa prática para que clientes externos participem, a experiência ajudou o laboratório a ser mais realístico, pois alunos agora lidam com clientes que não são da área de ciência da computação. Além disso, agora é possível para alunos que já cursaram a disciplina cursá-la novamente, especializando-se no papel de treinador.

Como neste contexto os alunos passam menos tempo no laboratório do que num contexto industrial, algumas práticas tiveram que ser adaptadas. O jogo do planejamento dos diferentes projetos tenta separar *releases* com poucas histórias (de 5 a 10), que devem ser implementadas em até quatro semanas. Em muitos casos tivemos que utilizar clientes *proxy* pois os verdadeiros clientes não podiam

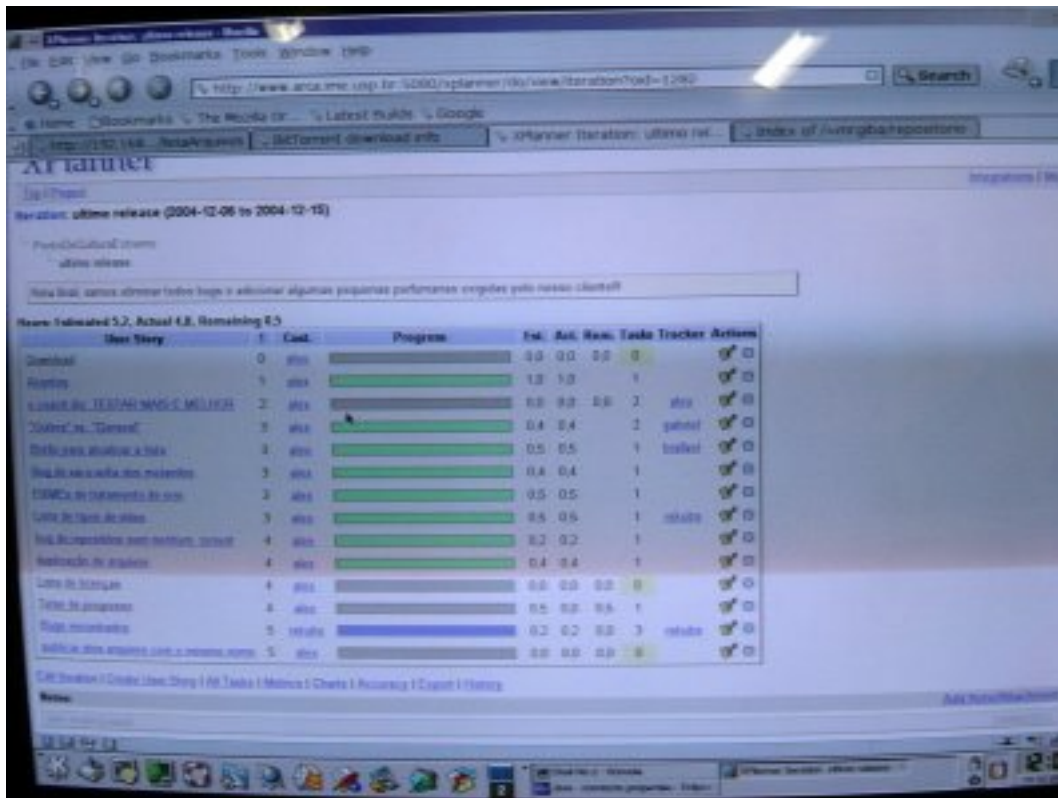


Figura 3.7: Histórias de uma equipe on-line no XPlanner.

comparecer a todas as aulas, tentávamos fazer com que os clientes reais participassem de jogos do planejamento e de apresentações dos *releases*, a Figura 3.8 mostra a apresentação de um *release* do Borboleta aos seus clientes. O ritmo sustentável foi estabelecido para refletir as 10 horas sugeridas de trabalho semanal. As outras práticas foram utilizadas sem maiores adaptações. Além das práticas comuns, as equipes tem a liberdade de adotar mais práticas ágeis ao seu contexto. Sugerimos sempre o uso de papos-em-pé e retrospectivas, que nos últimos anos tem sido bastante utilizadas.

A seguir iremos descrever experiências específicas em 4 projetos diversos desta disciplina.

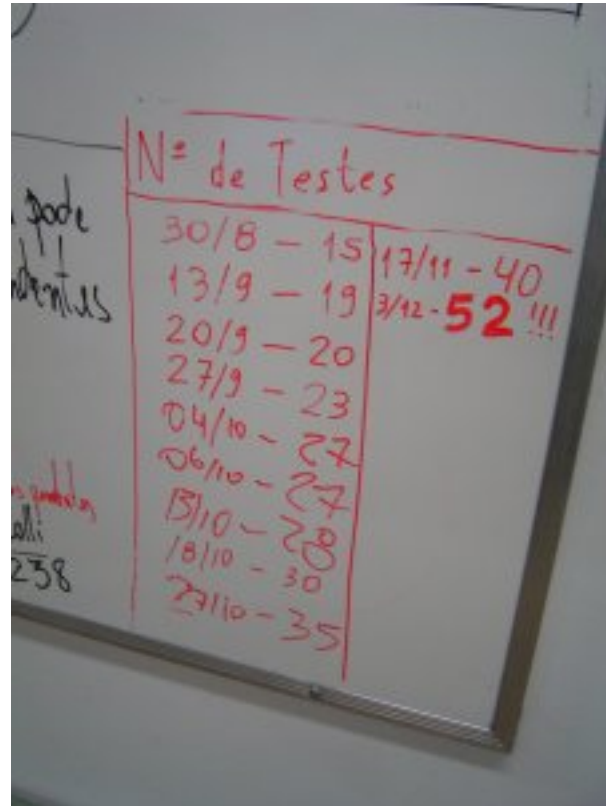


Figura 3.8: Enfermeiras clientes do Borboleta acompanham a apresentação de um release.

### 3.2.2 Mico - Sistema para administração de carga didática - nossa primeira experiência

Eu passava pela minha primeira experiência profissional real, desenvolvendo sistemas para uma multi-nacional na Itália, na qual durante um ano lidava com a frustração de ser um dos programadores mais qualificados da equipe e ao mesmo tempo ter que atuar politicamente na empresa e tentar coordenar o desenvolvimento de vários programadores trabalhando em três sistemas diferentes. Foi quando recebi o e-mail convite dos professores para participar de uma nova disciplina optativa no IME/USP, o Laboratório de Programação eXtrema. Ao ler as referências, me encantei com as soluções propostas aos problemas reais que estava enfrentado, e resolvi voltar ao Brasil para terminar meu curso.

Na primeira instância da disciplina, 12 alunos se dividiram em dois grupos e se dedicaram a



A whiteboard with a red border and title 'Nº de Testes'. The board is divided into two columns by a vertical red line. The left column lists dates and test counts: 30/8 - 15, 13/9 - 19, 20/9 - 20, 27/9 - 23, 04/10 - 27, 06/10 - 27, 15/10 - 28, 18/10 - 30, 27/10 - 35. The right column lists: 17/11 - 40, 3/12 - 52!!!. To the left of the table, there are some handwritten notes in black and red ink, including 'pode', 'dentis', and '238'.

| Nº de Testes |              |
|--------------|--------------|
| 30/8 - 15    | 17/11 - 40   |
| 13/9 - 19    | 3/12 - 52!!! |
| 20/9 - 20    |              |
| 27/9 - 23    |              |
| 04/10 - 27   |              |
| 06/10 - 27   |              |
| 15/10 - 28   |              |
| 18/10 - 30   |              |
| 27/10 - 35   |              |

Figura 3.9: Evolução de número de testes de uma equipe detalhada numa tabela pelo seu acompanhador.

resolver o mesmo problema, desenvolver um sistema para administração de carga didática do departamento de Computação. Na sua primeira experiência de ensino de XP, três professores se revezavam nos papéis de treinadores e clientes de ambas equipes.

Uma das tarefas administrativas principais de escolas e universidades é a construção do cronograma de aulas e conseqüente alocação de professores aos cursos que devem lecionar em um período. Muitas restrições são importantes nessa alocação: alunos têm preferências de disciplinas optativas e períodos nos quais elas devem ser oferecidas, professores tem suas próprias preferências de cursos e horários e o departamento deve satisfazer suas necessidades curriculares. Achar um cronograma e alocar professores é uma tarefa complexa que o departamento encara todos os anos. Por não conhecer nenhum software que resolvia este problema, os professores sugeriram este como o primeiro desafio

do laboratório. Assim como os alunos, eles seriam usuários do sistema, devendo indicar suas preferências e se beneficiariam de sua existência, pois o problema da carga didática todo ano incomodava o departamento.

A proposta era criar um sistema baseado na Web onde alunos e professores indicariam suas preferências para o ano letivo seguinte, professores diriam qual horário e dias da semana estariam disponíveis para aulas, além de quais disciplinas tinham competência para ensinar e alunos escolheriam quais cursos gostariam de participar e votariam em disciplinas optativas que gostariam que fossem oferecidas. A aplicação então usaria um algoritmo genético para achar um cronograma otimizado e alocar professores aos cursos que deveriam lecionar.

Ambas equipes se dedicaram ao mesmo problema, porém optaram por usar tecnologias diferentes, e durante os jogos de planejamento negociaram as prioridades de histórias de maneira diferente com seus clientes. Enquanto uma equipe deu prioridade ao desenvolvimento do algoritmo genético, usando C++, outra começou desenvolvendo a interface Web e optou por Java e arcabouços livres disponíveis para esta linguagem.

Nesta instância da disciplina ambas equipes tiveram problemas com testes automatizados, uma porque o algoritmo a ser testado era muito complexo, e a outra por ainda não existirem arcabouços de testes automatizados para interfaces Web. Além disso notamos que os professores tinham dificuldade em separar bem as preocupações que tinham como clientes ou treinadores, pois exerciam, ao mesmo tempo, ambos papéis. Curiosamente, a adoção do algoritmo genético foi importante para criação de uma metáfora comum para o desenvolvimento.

Ao final da disciplina, uma equipe conseguiu entregar uma versão quase funcional do algoritmo genético, e a outra uma interface Web completamente funcional, mas que só armazenava os dados necessários e não resolvia o problema. Após a disciplina, dois alunos continuaram o desenvolvimento do sistema, tentando integrar o algoritmo genético a interface Web sem sucesso, e posteriormente tentando implementar um novo algoritmo, também sem sucesso. Esta experiência foi relatada por nós neste artigo: [54]. Mesmo sem o algoritmo funcional, o sistema foi colocado em produção e usado durante quatro anos para coletar preferências dos alunos e professores para posterior criação da carga didática do departamento.

### 3.2.3 Marcador de Reuniões - grupo pequeno utiliza Smalltalk

Como auxiliar de ensino desta instância da disciplina, ajudei na avaliação e acompanhamento das equipes. A primeira delas, composta por 3 alunos e um professor que atuava como cliente, treinador, e também desenvolvedor, desenvolveu uma aplicação Web para marcar reuniões.

O projeto surgiu da necessidade dos professores do IME-USP de uma ferramenta para gerenciar suas reuniões. Um usuário cadastrado no sistema pode criar uma reunião, indicando em quais datas e horários ela poderia ocorrer e convidando outros usuários. Cada convidado pode indicar quais são os horários que ele prefere, o sistema então mostra ao usuário que convocou a reunião qual o melhor horário, ele seleciona sua preferência e pode também reservar uma sala.

A equipe começou com uma base de código existente, resultado do trabalho final da disciplina de programação orientada a objetos oferecido no semestre anterior, na qual também fui auxiliar de ensino. Os alunos escolheram entre 3 projetos em Smalltalk e optaram por dar continuidade ao projeto que tinha a interface mais funcional. Os sistemas já contavam com uma base de testes automatizados, que havia sido desenvolvida para ajudar na avaliação da disciplina de orientação a objetos. Optou-se pelo VisualWorks Smalltalk<sup>1</sup> em conjunto com os arcabouços SUnit para testes, VisualWave como servidor Web e Store para controle de versões e integração contínua.

Notamos que grande parte do sistema foi refatorada, e no final somente os testes e aspectos da interface permaneceram em uso. A presença do cliente se deu em todas as aulas, pois o professor que atuava como desenvolvedor e treinador também era cliente, porém como a equipe era pequena, não observamos problemas com essa prática.

### 3.2.4 Colméia - Gerenciador de Biblioteca - evolução de um projeto ao longo dos anos

Dando continuidade ao sistema Colméia, iniciado na instância anterior da disciplina, esta equipe contava com 8 alunos, sendo que um já havia cursado a disciplina anteriormente e atuava como treinador. Junto com o código fonte que já estava pronto, os alunos herdaram também um cliente, já com alguma experiência em participar de projetos XP. O cliente não podia estar presente em todas as aulas, mas participou de jogos do planejamento e avaliação de *releases*.

O sistema já contava com funcionalidades de controle de consulta e empréstimo de periódicos,

---

<sup>1</sup>atualmente na disciplina de programação orientada a objetos do IME/USP não mais utilizamos o VisualWorks e sim Squeak Smalltalk, versão completamente livre em conjunto com o arcabouço Seaside para geração de páginas web.

livros e demais materiais. Nesta edição os alunos deveriam desenvolver o módulo de controle de pessoas do sistema.

No início do projeto, os alunos se dedicaram a conhecer o sistema existente e as tecnologias adotadas. O fato de uma base de código existir foi positivo, os alunos conseguiram adotar rapidamente um ritmo eficiente de desenvolvimento e ressaltaram a importância do código fonte dos testes no aprendizado do sistema.

Vale notar que este grupo passou por uma experiência peculiar, quando o aluno que atuava como treinador teve que abandonar a disciplina no meio do semestre para se dedicar a uma oportunidade profissional. A solução a este empasse surgiu quando outro aluno voluntariamente aceitou o papel de treinador. Com ajuda da sua equipe, se apoiaram no uso de todas as práticas de XP e implantaram a metodologia de maneira eficiente, mesmo essa sendo sua primeira experiência.

### **3.2.5 Cigarra - Distribuidor Multimídia - clientes externos em um grande projeto governamental**

A maior equipe neste semestre, com 11 alunos, se dedicou a implementar um projeto proposto em parceria com o Ministério da Cultura. Eu atuei como treinador desta equipe, e também era consultor no Ministério, cliente *proxy* e atuava como desenvolvedor. O cigarra, como foi chamado, seria utilizado pelo projeto “Pontos de Cultura” do Ministério. Acreditamos que os alunos ficaram motivados a participar deste projeto devido ao fato de existirem clientes reais fora do IME/USP. A figura 3.10 mostra a atuação do treinador desta equipe.

A idéia do projeto “Pontos de Cultura” era apoiar projetos culturais com ação prévia comprovada em áreas remotas e periféricas do país com financiamento anual e um *kit* de produção multimídia, composto por dispositivos de produção de áudio e vídeo, microfones e câmeras, uma conexão de banda larga à Internet e um servidor pré-configurado para compartilhamento da rede. Pessoas de comunidades ligadas aos pontos de cultura seriam estimuladas a usar estes equipamentos e produzir músicas, filmes e outros artefatos licenciados de maneira a facilitar a distribuição do conteúdo na Internet. Todo este conteúdo cultural, em forma de vídeo, áudio, fotos e textos, deve ser publicado e distribuído para qualquer pessoa interessada, em qualquer lugar do mundo.

A sugestão foi criar um sistema *peer-to-peer* de troca de arquivos que considerasse o problema de publicação de dados e busca por informações. Dois requisitos eram importantes, o sistema deveria ser muito fácil de usar e não poderia ter problemas em escalar, pois a previsão é que a rede de usuários



Figura 3.10: O treinador da equipe Cigarra atuando.

publicando e acessando material poderia crescer muito rapidamente.

Durante a primeira iteração da equipe, pesquisas foram realizadas para escolha da tecnologia a ser adotada, e o protocolo *BitTorrent* se mostrou o mais adequado, junto com a tecnologia de *rich-client-application* da plataforma Eclipse e o software Azureus.

O sistema foi desenvolvido em 4 *releases*, sendo que alguns *releases* foram testados por outro consultor do Ministério, para que eu não precisasse atuar constantemente como treinador e também cliente *proxy*. Tivemos dificuldades com testes automatizados, por mais que a acompanhadora sempre indicava a falta de testes a equipe ficou muito ansiosa para terminar o desenvolvimento e não testou o software de maneira consistente, este fato se refletiu na quantidade considerável de defeitos encontrados após cada *release*. Na última semana de desenvolvimento conseguimos chegar a um nível

de testes aceitável, que cobria toda a aplicação. No seu último *release* a aplicação contava com 3695 linhas de código, sendo 2995 linhas relativas à aplicação (27 classes) e 700 relativas aos testes (17 classes). Detalhes mais técnicos sobre o sistema desenvolvido podem ser encontrados em artigo por nós publicado no sexto Workshop sobre Software Livre do FISL 2005 [53].

Todas as histórias mais prioritárias do cliente foram implementadas, mas infelizmente, após o término da disciplina, o Ministério não colocou o sistema em produção.

Vale notar que nesta equipe utilizamos o software *XPlanner* para armazenar todas as histórias *on-line* e ainda atualizar dados sobre pareamentos e tempo gasto no desenvolvimento.

Notamos ainda que por mais que o desenvolvimento começou sem base de código, utilizar *softwares* livres como ponto de partida se mostrou uma boa opção. Reutilizamos muito código, e tivemos a oportunidade de aperfeiçoar técnicas de testes automatizados escrevendo testes para código livre, assim contribuindo um pouco com a comunidade. Em comparação com as outras equipes desta instância da disciplina, a equipe do Cigarra sentiu que não ter uma base de código para começar o desenvolvimento os colocou em desvantagem, atrasando o primeiro *release*. A figura 3.11 mostra a reunião de apresentação do primeiro *release* da equipe.

Além disso, a equipe era grande, 12 pessoas no total, limite inicialmente proposto por Beck para fazer XP eficientemente e por isso sentimos que a comunicação foi um pouco falha. Como treinador tentei ao máximo melhorar a comunicação, porém o pouco comprometimento de alguns alunos que faltaram muito, complicou este desafio. Uma nova prática proposta foi a de fazer com que os jogos do planejamento tivessem duas fases, uma presencial que durava no máximo uma hora, e outra que continuava até a próxima aula e acontecia na lista de discussão por e-mail da equipe. Esta prática otimizou o tempo disponível para parear presencialmente. Outra prática adotada foi a de dar “créditos presença” para pares que investissem seu tempo em tarefas prioritárias nos últimos *releases*. Desta maneira os alunos tinham um incentivo a mais para parear fora do horário de aula e repor créditos que perderam devido a faltas.

### 3.3 Trabalhos relacionados na Indústria

Muitas empresas não utilizam nenhum método definido para o desenvolvimento de software, ainda mais em economias em desenvolvimento como no Brasil, deixando a programação de sistemas à mercê de seus desenvolvedores, que adotam práticas *ad hoc*. Para melhorar a qualidade do seu trabalho, e ter mais controle sobre os seus processos de produção, algumas empresas optam por investir em



Figura 3.11: Reunião de apresentação do primeiro *release* da equipe Cigarra.

métodos de desenvolvimento. Existem muitas opções, aquelas que podem arcar com os custos optam por alternativas que oferecem certificações reconhecidas no mercado e adotam métodos tradicionais ou processos oferecidos por consultorias especializadas. Algumas reconhecem que métodos ágeis, por mais jovens que sejam, oferecem uma alternativa viável.

O ensino de XP na Indústria pode ocorrer através de consultorias, onde um treinador é contratado para trabalhar junto com a equipe que vai adotar a metodologia. Nestes casos, o treinador permanece com a equipe até o momento no qual esta pode continuar sozinha, tendo aprendido não só as práticas, mas também como adapta-las e evoluí-las no seu contexto. Porém, todas empresas são diferentes, de forma que a adoção de XP em uma empresa sempre varia de acordo com a realidade local, podendo ser feita inclusive sem auxílio de um treinador experiente. Muitas empresas adotam XP após alguma pessoa da empresa entrar em contato com o método e convencer seus colegas (e a

gerência) a experimenta-la.

Em seu trabalho de mestrado, Vinicius [147] relata um caso de sucesso de adoção de Programação eXtrema como método em sua empresa no Brasil. Neste caso, parte de um sistema comercial foi construído durante 14 meses por 4 desenvolvedores experientes, usando as práticas e valores de XP e plataforma Java J2EE, com auxílio de um dos sócios da empresa como treinador. O cliente era um consórcio de empresas que ganhou uma concorrência pública e aceitou um contrato de escopo negociável para o projeto. Entre todas as práticas, somente testes automatizados não foi adotada de maneira eficiente. Programação pareada obteve resistência dos desenvolvedores somente no início do projeto. Após 3 iterações (que duravam 2 semanas) em que os desenvolvedores concordaram em adotar todas práticas de XP, ela foi aceita como uma boa prática e mantida. A prática de testes automatizados não foi efetiva, sendo que cada *release* apresentou uma média de 10 defeitos por iteração. A falha nesta prática foi amortizada pelo trabalho de um analista de testes que identificava e apontava falhas após cada *release*. O estudo conclui que XP foi aplicada com sucesso, porém não pode apontar se este caso foi mais eficaz do que seria usando outros métodos, pois o projeto demorou mais tempo e custou mais caro do que o previsto inicialmente. Isto se justifica dado ao surgimento de muitas novas funcionalidades e mudanças conceituais ao longo do desenvolvimento, quando *releases* eram apresentados aos clientes. O resultado foi uma grande mudança no escopo da aplicação devido ao aprendizado dos clientes ao longo do projeto. Ressalta-se porém que os clientes ficaram satisfeitos com o sistema ao final do processo, e este, ao ser colocado em produção (o sistema seria utilizado de maneira sazonal, anualmente), apresentou somente 3 defeitos, sendo que o primeiro defeito só foi encontrado 4 meses e meio após o sistema começar a ser utilizado, nota-se que *feedback* veloz é essencial para ajustar devidamente o escopo do projeto e obter satisfação dos clientes e o uso de refatoração é importante para evoluir o *design* simples.

Em uma experiência semelhante, Pelrine [119] relata um caso de sucesso no qual uma empresa de consultoria na Suíça também participou de uma concorrência e teve que adaptar XP a um contrato de preço fixo. O autor reflete sobre o alto risco envolvido neste tipo de contrato devido a mudanças não previstas de escopo e ressalta que, por ser mais flexível, XP pode ser uma boa opção e por isso resolveram adotá-la. O cliente era do ramo farmacêutico e não estava satisfeito com a qualidade do software que obtinha de fornecedores externos até então, por isso concordou em usar XP. A análise de requisitos foi feita como um jogo do planejamento, o que permitiu aos desenvolvedores estimar o tempo envolvido com o trabalho e especificar um preço para a concorrência. O resultado foi que XP se mostrou tão ágil que a equipe de desenvolvimento foi até forçada a tirar férias enquanto o

cliente obtinha tempo para poder validar um *release*. Todas as práticas de XP foram seguidas à risca, porém, devido a resistência da gerência, programação pareada não foi adotada em um pouco mais de 50% da produção de código. A equipe era de apenas um desenvolvedor e este envolvia outro para revisões de código, tentando fazer programação pareada de “guerrilha”. Quando isso não era possível, o desenvolvedor pareava com o cliente, este ressalta que o uso de Smalltalk foi importante para o desenvolvimento de uma metáfora comum e a participação efetiva do cliente. O projeto foi um sucesso, sendo entregue com o custo previsto antes da data final estimada.

Deias et al. [42] relatam sua experiência de adotar XP em uma empresa especializada no desenvolvimento de portais complexos para clientes externos na Itália. A empresa utilizava ferramentas e processos do RUP<sup>2</sup> e procurava obter uma certificação ISO 9001 de qualidade para o seu processo de desenvolvimento quando optou por adotar XP. Durante 2 anos experimentaram XP em dois projetos pilotos para melhorar a qualidade do software produzido e reduzir problemas com a documentação, muito grande e freqüentemente desatualizada, gerada pelos seus processos. A implantação de XP começou com entrevistas com funcionários de diversos departamentos, estes deveriam relatar problemas que estavam enfrentando e então participar de discussões sobre as práticas de XP e como elas poderiam ajudar. Resistência surgiu principalmente das áreas de marketing e recursos humanos, em especial os funcionários responsáveis pelo fechamento de negócios não sabiam como poderiam vender contratos de escopo negociável a clientes acostumados ao escopo, custo e data de um projeto serem sempre fixos. Funcionários da área de RH tinham medo da pratica do cliente sempre presente por achar que desta maneira iriam expor o funcionamento interno da empresa, criando visibilidade demais. Durante os dois projetos piloto, equipes de 6 a 7 desenvolvedores trabalharam durante 3 meses com o cliente sendo representado pelo diretor do laboratório de P&D, sendo que 5 pessoas da equipe tinham menos do que 2 anos de experiência com programação. O primeiro projeto foi desenvolvido em C++ e as práticas de jogo do planejamento e padrão de codificação não foram adotadas, o planejamento seguiu práticas como planilhas de GANTT sugeridas pelo RUP. O segundo projeto foi desenvolvido em Java, seguindo todas as práticas de XP, inclusive iterações de duas semanas, sessões de desenvolvimento de meia hora, e *releases* a cada duas iterações. Os autores relatam que adotar XP parcialmente não foi satisfatório, misturar técnicas de planejamento do RUP não foi eficiente e que perceberam os benefícios da sinergia entre as práticas somente ao adotar XP como um todo. Ressaltam dificuldades com refatoração e uso de metáfora devido à inexperiência da maioria dos desenvolvedores, o que também fez com que o treinador tivesse um papel importante durante

---

<sup>2</sup>Rational Unified Process *TODO AQUI referência seminal.*

jogos do planejamento para incentivar a participação de pessoas menos experientes. Em seu segundo piloto notam que o jogo do planejamento é mais eficiente do que planejamento RUP e melhora a percepção da equipe quanto ao real andamento do projeto, aumentando o nível de motivação dos desenvolvedores. Ressaltam as dificuldades sofridas pela inexperiência da equipe e resistência política devido a capacidade de XP de nivelar o conhecimento de programadores em um ambiente no qual a progressão de carreira projetava programadores a analistas de requisitos e depois a gerentes de projeto. Relatam a importância de adotar XP como um todo e de utilizar um cliente *proxy* ativo quando a presença de clientes dentro da empresa não é desejável. Concluem dizendo que XP é mais fácil de adaptar a um arcabouço ISO 9001, sendo mais ágil do que outros métodos para obter este tipo de certificação. Atualmente 15 desenvolvedores trabalham na empresa utilizando XP.

Já Howard [71] relata um caso de adoção similar, no entanto, em uma empresa americana que utilizava o modelo em cascata. Uma equipe liderada por um desenvolvedor com pouca experiência prévia com XP aproveitou a insatisfação de um cliente, que ameaçou cancelar o contrato da segunda fase de um projeto devido ao alto custo e tempo levado na primeira fase, para convencer a gerência a experimentar XP. O autor aponta muitas dificuldades neste contexto, onde uma equipe de 12 pessoas (algumas que seguiam o processo do modelo em cascata, 3 delas trabalhando *offshore* na Índia) enfrentou problemas para aprender as práticas de XP, conviveu com falhas de comunicação, resistência aos testes, integração contínua e ritmo sustentável. Conclui apontando que mesmo assim o projeto foi efetivo e satisfaz o cliente principalmente devido aos *releases* frequentes e possibilidade de re-planejamento. Aponta que após algum tempo a equipe que trabalhava colocada conseguiu estabelecer um bom ritmo de desenvolvimento que se adaptava bem a mudanças e que em uma próxima oportunidade tentarão investir mais no aprendizado da equipe e melhor integração com o resto da empresa.

Em uma comparação entre dois *releases* de um mesmo projeto, Layman et al. [86] falam da experiência da empresa aérea Sabre nos Estados Unidos. No primeiro *release* o modelo de cascata era utilizado com algumas práticas de XP como integração contínua, no segundo XP foi adotada completamente. A equipe foi uma das primeiras a adotar XP dentro de empresa e contava com 10 desenvolvedores. O programa sendo desenvolvido atendia as necessidades de aproximadamente 30 clientes externos. Os autores notam que a equipe era caracteristicamente ágil e que contava com apoio da gerência e da organização para adotar XP. O estudo mostra um aumento de 50% em produtividade, 65% em qualidade pré-release e 35% de qualidade pós-release. A única prática que sofreu resistência da equipe foi programação pareada, desenvolvedores não viam valor em aplicá-la

em tarefas consideradas simples, a prática foi utilizada somente 50% do tempo. Atualmente 200 desenvolvedores da Sabre utilizam XP, distribuídos em mais de 30 equipes.

Fuqua e Hammer [60] relatam um caso típico onde uma equipe XP teve que se adaptar a muitas mudanças. No decorrer deste projeto interno em uma empresa que desenvolvia sistemas de segurança para a Internet nos EUA, de 2 a 4 desenvolvedores tiveram que lidar com mudanças de treinador, clientes, da própria equipe, requisitos e até a parada e retomada do projeto. A equipe contava com 30 clientes internos e trabalhou cerca de 40 iterações de duas semanas em 252 histórias. Os autores ressaltam a importância de não tentar descobrir todas as histórias previamente (durante uma fase de planejamento inicial identificaram menos de 50% das histórias) e também não se preocupar em descartar histórias (8% foram descartadas). Relatam também a importância de usar um Wiki para lidar com o grande número de clientes e pequenas histórias. Neste Wiki estimativas de tempo para completar blocos de histórias satisfaziam os clientes e eram usadas pela gerência quando surgia necessidade de re-priorizações. Quando surgiram novos clientes relatam que o *design* simples facilitou a adaptação do projeto aos novos requisitos. Quando a mesma equipe teve que assumir um outro projeto em paralelo, mostram que bastou incorporar histórias de ambos projetos em suas iterações para lidar com a nova responsabilidade. Parar e retomar o projeto foi simples, bastando para isso parar após o final de uma iteração, ressaltam a importância da documentação de histórias no Wiki e propõem que uma semana de refatoração é uma excelente maneira de retomar um projeto com rapidez. Quando a equipe teve que crescer, relatam que é importante escolher desenvolvedores motivados, não importando tanto suas habilidades técnicas e mostram como programação pareada foi importante para que os novos desenvolvedores aprendessem XP e já contribuíssem efetivamente no seu primeiro dia de trabalho. Com a saída de um desenvolvedor, perceberam que a prática de propriedade coletiva de código não tinha sido tão eficiente, pois tiveram dificuldades de dar continuidade ao desenvolvimento de certas partes do sistema. A saída do treinador, no entanto, foi tranquila, outro desenvolvedor assumiu as responsabilidades sem grandes problemas, mostrando que o trabalho do treinador foi realmente bem feito. Concluem sua análise ressaltando a importância de um bom treinador, uma equipe motivada, e muita programação pareada para o sucesso deste projeto repleto de mudanças.

O caso de uma empresa de integração de sistemas Italiana desenvolvendo uma aplicação de acompanhamento de portfólios para investidores de um banco privado relatado por Bossi [24] é exemplar. Nele, o banco aceitou um contrato de escopo negociável e disponibilizou um cliente proxy para trabalhar durante 3 dias por semana co-locado com a equipe de 4 a 6 desenvolvedores da

empresa. Além deste cliente proxy e dos desenvolvedores, um treinador e um gerente de negócios da empresa relacionavam-se com um gerente do banco, que tomava as últimas decisões sobre mudanças de escopo do projeto. Membros da equipe revezavam-se no papel de acompanhador gastando 15 minutos diários para atualizar métricas, o que garantia *feedback* efetivo. Todas as práticas foram seguidas à risca e além delas a equipe se beneficiou de um Wiki e planilhas de acompanhamento. A equipe evoluiu o seu uso de XP com novas práticas como papos-em-pé aliados a diários no Wiki, que relatavam os resultados de cada sessão de programação pareada. Estas práticas, em conjunto com retrospectivas e tempo dedicado a pesquisa, garantiam uma cultura de aprendizagem contínua que está explícita na evolução das estimativas: em média a equipe subestimou suas histórias em 15,68%, mas nos últimos 3 meses esse erro caiu para 0,61%. Bossi relata que o cliente ficou muito satisfeito com o contrato de escopo negociável quando percebeu que um requisito de obter informações em tempo-real não poderia ser desenvolvido devido a limitações de outro de seus fornecedores e pode adaptar o plano, satisfazendo-se com informações atualizadas a cada 15 segundos. O projeto durou 10 meses e meio e contou com 26 iterações, 12393 linhas de código executável foram produzidas num sistema com 297 classes, 886 testes de unidade e 161 testes de aceitação. Somente 14 defeitos foram encontrados, sendo que apenas 3 após o sistema entrar em produção. Considerado seu relato um sucesso total, o autor nota que o jogo do planejamento, acompanhamento efetivo, retrospectivas e a presença constante do cliente foram essenciais para obter confiança mútua e alterar devidamente o escopo do projeto, garantindo assim o seu êxito.

Na próxima seção descreveremos em detalhes nossa experiência ensinando e adaptando XP a uma empresa *start-up* no Brasil [55].

### 3.4 Paggo - uma *start-up* brasileira adota XP

Aprender a adaptar-se a mudanças é especialmente importante em uma economia em desenvolvimento como a Brasileira na qual empresas abrem e fecham rapidamente, flutuações econômicas são frequentes, bons desenvolvedores são difíceis de encontrar e muitos negócios sobrevivem reciclando estagiários constantemente. Salários baixos são a norma e refletem uma força de trabalho desqualificada e uma cultura de contínua troca de funcionários. Ferramentas e arcabouços tem pouca documentação em português. Desenvolver *software* de alta qualidade, em tempo e no orçamento é requisito básico se uma empresa pretende sobreviver neste contexto. Para tanto, eu fui contratado para ajudar uma empresa *star-up*, a Paggo, a adotar XP como metodologia. A Paggo entrava no negócio altamente competitivo de cartões de crédito, apostando em um novo modelo de negócios

baseado em novas tecnologias e utilização de um método ágil para conseguir ter *software* funcional rapidamente, reduzindo o tempo de entrada no mercado com objetivo de obter mais investimentos. A figura 3.12 mostra 4 desenvolvedores pareando no escritório da Paggo.



Figura 3.12: Dois pares trabalhando na Paggo.

Desde o início, muitos desafios se apresentaram; acreditávamos que o mais difícil seria lidar com o aspecto heterogêneo da equipe, composta de desenvolvedores com diferentes habilidades: de estagiários com pouca ou nenhuma experiência programando à *seniors* acostumados ao sua própria maneira de programar. O fato que eu, como treinador, não poderia estar presente em tempo integral devido ao orçamento limitado também foi visto como um desafio. Mesmo assim tínhamos esperança já que adotar XP tinha sido sugestão de um membro da equipe e toda a equipe aceitou o desafio.

Nosso objetivo principal era ter uma equipe proficiente em XP e pronta para trabalhar independente do treinador em 6 meses, nos quais, além de treinar a equipe nas práticas de programação

extrema, também ensinávamos tecnologias e até mesmo aspectos de orientação a objetos,

O sistema a ser desenvolvido era altamente inovador, usando tecnologias como Java J2EE e J2ME e arcabouços livres e de código aberto como *VRaptor*, *JBoss* e *Hibernate*. O projeto tinha muitos aspectos, desde a gerência de transições de crédito com requisitos de alto desempenho até tecnologia móvel embarcada em telefones celulares e um portal dinâmico onde clientes poderiam pedir cartões e verificar seu balanço mensal.

A equipe era heterogênea, composta inicialmente por 6 desenvolvedores, habilidades variando de estagiários com quase nenhuma experiência em programação ou conhecimentos de orientação a objetos até desenvolvedores com muitos anos de experiência. Eu atuava como treinador, presente em tempo parcial, comparecendo a empresa pelo menos 3 dias da semana. O papel do cliente era desempenhado por um dos fundadores da empresa, presente durante todo o tempo. Outro consultor em tempo parcial ajudou a treinar a equipe no uso das tecnologias necessárias durante os primeiros meses. O papel de acompanhadora era realizado por uma estagiária que se sentiu muito motivada ao exercê-lo, a Figura 3.13 mostra a acompanhadora em ação. No início do quarto mês mais dois desenvolvedores foram contratados, contribuindo para código colocado em produção logo na primeira semana de trabalho, devido a prática efetiva de programação pareada e o fato da equipe já estar confortável com XP, além de um deles ter participado do laboratório de programação extrema no IME/USP.

Acreditávamos que o maior desafio seria lidar com a heterogeneidade da equipe, como incentivar todos a seguirem XP mas ao mesmo tempo lidar com dificuldades individuais? Mesmo tendo toda equipe dedicada a aprender XP, os desenvolvedores mais experientes resistiam a algumas práticas como programação pareada, propriedade coletiva do código e integração contínua, em especial o desenvolvedor mais velho da equipe se mostrou altamente resistente a mudar seus hábitos. Além disso, muitos não tinham as habilidades necessárias para praticar testes automatizados, *design* simples e refatoração.

Decidimos ensinar XP com todas práticas desde o início, mesmo cientes de que demoraríamos algum tempo para alcançar níveis maduros e aceitáveis em algumas delas. Durante 6 meses fizemos 12 *releases* em sua maioria desenvolvidos em iterações de duas semanas. Produzimos 4 aplicações, implementando com sucesso 269 histórias de um total de 340 escritas pelo cliente, destas 42 foram descartadas ou consideradas desnecessárias por este. De um ponto de vista técnico, entregamos 90% das funcionalidades desejadas, todas testadas e sem defeitos. A Figura 3.14 mostra o radiador de



Figura 3.13: Acompanhadora atualizando radiadores de informação Paggo.

informação de acompanhamento de um *release*. Do ponto de vista do negócio o projeto foi considerado um sucesso total, hoje a Paggo se estabelece como líder no mercado brasileiro.

Durante os primeiros 2 meses exploramos todas as práticas de XP, mas com muita calma na adoção de práticas que demandavam mais conhecimentos técnicos como desenvolvimento orientado a testes, integração contínua e refatorações. Nos próximos 4 meses treinamos a equipe em padrões orientados a objetos e nos arcabouços de código aberto sendo utilizados. Ao mesmo tempo que a equipe ficou mais confortável com padrões e técnicas mais avançadas de orientação a objetos, as práticas de testes e refatoração evoluíram.

Após participar de uma conferência sobre XP, o treinador introduziu algumas novas práticas, sendo retrospectivas a mais importante e mais efetiva. Usávamos um quadro de histórias para



Figura 3.14: Radiador de informação com acompanhamento de um dos primeiros *releases* de sistemas da Paggo.

acompanhar o progresso do projeto, onde passamos a coletar informações diariamente para usar em retrospectivas freqüentes, a Figura 3.15 mostra o quadro de histórias. Além do quadro a equipe contava com um espaço aberto e fazia uso de vários radiadores de informação, a Figura 3.16 mostra um radiador de informação utilizado.

A presença do cliente foi muito importante, tanto para re-priorizar histórias quanto para motivar a equipe na transição para XP, o cliente era também o chefe e apoiou completamente a adoção da metodologia. Sua participação foi essencial no início do projeto, onde contribuía até mesmo nos papos-em-pé, verificando o andamento da iteração e adaptando o escopo dos primeiros *releases* constantemente devido a estimativas erradas em razão da pouca prática da equipe. Seu papel também se mostrou essencial quando chegamos a conclusão que era melhor que o desenvolvedor mais resistente

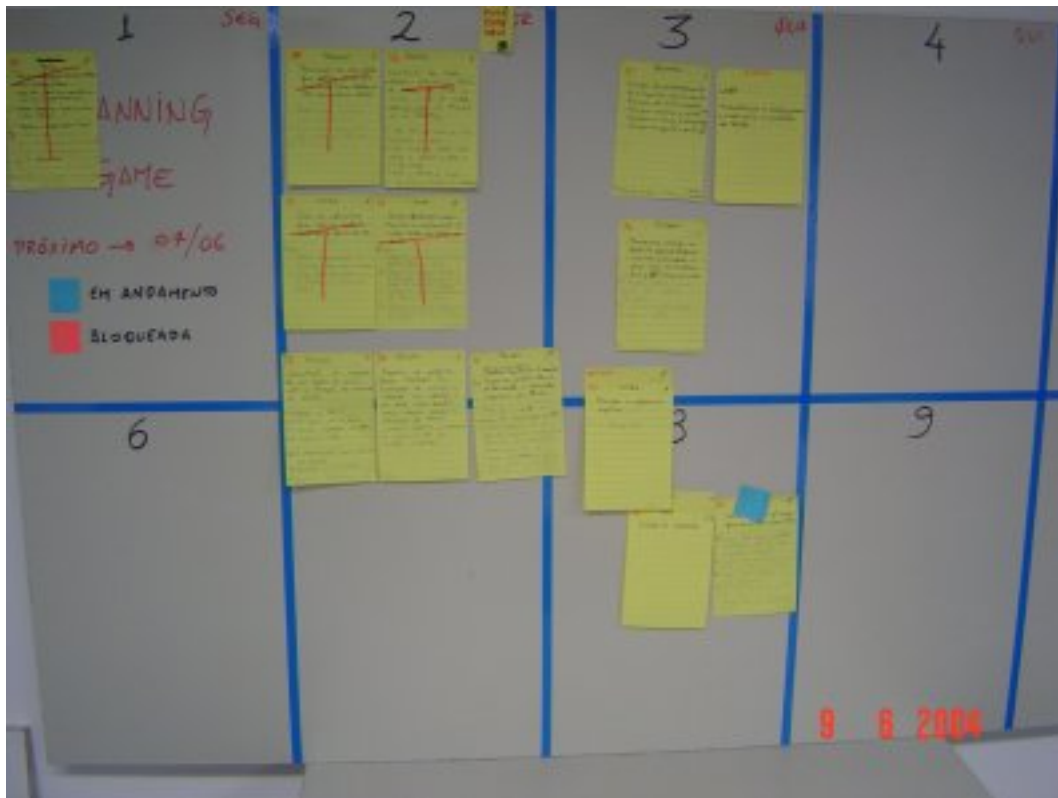


Figura 3.15: Quadro de histórias para acompanhamento de um *release* na Paggo.

a adoção de XP deixasse a equipe, tomar essa decisão foi importante para manter a equipe engajada no aprendizado do método.

No final do quinto mês a empresa teve que cortar custos pois não tinha ainda garantido novos investimentos, nesta época o treinador entrou em acordo com o cliente para dedicar seu último mês de trabalho para garantir que a equipe poderia continuar fazendo XP sem sua presença, o que aconteceu sem dificuldades.

Desde então a empresa obteve novos investimentos e mais do que duplicou sua equipe, práticas ágeis são usadas até mesmo em departamentos que não estão envolvidos com programação. Iremos discutir este papel de meta-treinador e o uso de práticas ágeis em equipes de outras áreas em mais detalhes nos próximos capítulos.

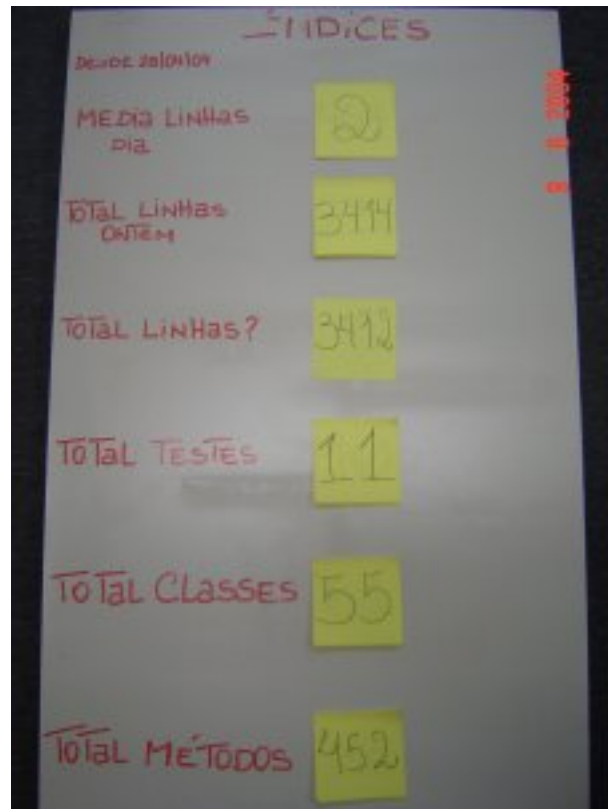


Figura 3.16: Radiador de informação mostrando evolução da base de código de um dos sistemas da Paggo.

### 3.5 Trabalhos relacionados no Governo

Poucos estudos existem sobre o ensino e adoção de XP dentro de contextos governamentais, talvez por que muitas vezes instituições governamentais delegam o trabalho de desenvolvimento de software para empresas contratadas por concorrências públicas, ou, quando têm equipes internas de desenvolvimento, estas adotam práticas guiadas por planejamento e documentação típicas de ambientes mais burocráticos. Porém observamos que métodos ágeis podem trazer benefícios mesmo nestes contextos onde mudanças acontecem, mas são implementadas com pouca velocidade e encontram muita resistência de funcionários públicos acostumados a burocracia da máquina governamental, que é uma barreira à aceitação de muitos dos princípios de XP.

O ensino de XP dentro de instituições do governo pode acontecer de diversas maneiras, sendo que

as iniciativas que observamos surgem através da motivação de alguns indivíduos dentro das equipes de tecnologia da informação e ocorrem muitas vezes por iniciativa própria dessas equipes, sem apoio institucional.

Em um estudo controlado, Wood e Kleb [164] relatam um caso onde uma equipe pequena (de 2 programadores) adotou XP sem problemas em um projeto piloto de missão crítica de um departamento de pesquisa da Nasa. Eles analisaram sua produtividade e a compararam com projetos passados, relatando que o uso de XP foi aproximadamente duas vezes mais produtivo. Ambos tinham pouca experiência com XP e aprenderam o método com apoio da literatura existente.

Alleman e Henderson. [3] relatam um caso de adoção parcial de XP onde adaptações tiveram de ser feitas ao acompanhamento do projeto para prover a documentação necessária para a burocracia governamental. Trabalharam no Departamento de Energia de Colorado nos Estados Unidos em um projeto crítico relacionado a armas nucleares, no qual o maior desafio era apresentar o andamento do projeto sob a forma de “valor produzido” a cada período de contabilidade. Isto os fez adaptar o acompanhamento de velocidade do projeto para o formato exigido, introduzindo custos de tempo tanto para o acompanhamento do projeto quanto para criação desta documentação extra. A equipe contava com aproximadamente 100 pessoas e os requisitos eram providenciados por um departamento inteiro responsável só por isso. Relatam dificuldades relacionadas aos relatórios financeiros, a necessidade de obter certificações CMMi<sup>3</sup> nível 4 e também a requisitos de segurança, que impossibilitaram a equipe de praticar programação pareada e propriedade coletiva do código (nem todos desenvolvedores tinham níveis de acesso de segurança para olhar partes do código). Concluem que é possível adotar XP a este contexto e obter benefícios das práticas utilizadas em conjunto com técnicas tradicionais para prover a documentação exigida pelos órgãos governamentais.

Yoder [63] relata um caso no Departamento de Saúde Pública de Illinois, Estados Unidos no qual sua empresa providenciou uma equipe para ajudar no desenvolvimento de um arcabouço que agregaria informações sobre dados comuns a aplicações médicas diversas, obtendo reutilização de código, criando maneiras mais fáceis e rápidas de entregar aplicações e compartilhando dados entre programas. A equipe de 10 pessoas utilizou muitas práticas de XP e foi treinada por Yoder que também atuou como arquiteto principal. O departamento utilizava cubículos para seus desenvolvedores e uma das primeiras coisas que fizeram neste caso foi criar um espaço aberto e comunicativo. O autor nota que o espaço aberto acabou criando visibilidade demais e os funcionários do departamento

---

<sup>3</sup>Capability Maturity Model *improved*. TODO REFERENCIA AQUI

não estavam confortáveis com a idéia. Críticas surgiram principalmente de outros funcionários do departamento, que ao ouvir conversas e interpretá-las erroneamente reclamavam para a gerência que os desenvolvedores estavam perdendo tempo. Sugere que teria sido melhor isolar o espaço aberto do resto do departamento. O autor nota a importância da prática de programação pareada quando pessoas saíam e entravam no projeto, facilitando a incorporação de novos desenvolvedores que se tornavam produtivos rapidamente. Porém nem todos desenvolvedores pareavam e algumas soluções individuais eram integradas (com testes e sem comprometer a propriedade coletiva do código), além disso os pares não se revezavam. Relatam também a importância dos testes como documentação de uso do arcabouço criado, e ressaltam a facilidade de criar testes de aceitação devido a adoção de Smalltalk. Dificuldades foram encontradas na prática de pequenos *releases*, por mais que, internamente, a equipe entregava *releases* com frequência, o departamento não via a necessidade de colocar estes em produção devido a percepção que seria uma perda de tempo colocar seus funcionários para utilizar um sistema que ainda estava em desenvolvimento. Notam que o maior problema encontrado foi na adoção de histórias, por não ter acesso aos funcionários que utilizariam o sistema, esta prática não foi adotada e os requisitos foram entregues por analistas do governo, fazendo com que um dos sistemas desenvolvidos gerasse reclamações dos usuários finais por não ser aquilo que precisavam. Concluem que a estrutura política de uma organização governamental foi o maior empecilho para adoção completa de XP.

Na próxima sessão iremos descrever em detalhes nossa experiência de alguns anos tentando implantar XP em um contexto governamental no Brasil.

### 3.6 Ministério da Cultura - Desenvolvimento de *software* no projeto Cultura Digital

No início de 2005 fui contratado para trabalhar como coordenador de tecnologia no Departamento de Cultura Digital do Ministério da Cultura Brasileiro. O projeto principal deste departamento estava ligado à distribuição de *kits* multimídia para cerca de 600 Pontos de Cultura distribuídos pelo país que seriam treinados no seu uso. Parte do trabalho técnico estava ligado ao desenvolvimento de três portais de suporte deste processo: o Conversê, no qual pessoas ligadas aos pontos poderiam se cadastrar, fazer amigos e conversar em uma rede social; o Mapsys, um sistema de gestão colaborativa da rede de pontos, no qual dados burocráticos e administrativos eram armazenados, atualizados e geo-referenciados; e o Estúdio Livre, plataforma na qual informações sobre o uso de *softwares* livres para produção multimídia seriam agregadas e artefatos digitais criados seriam disponibilizados em

um acervo por uma comunidade.

Inicialmente a equipe técnica dedicada ao desenvolvimento desta plataforma contava apenas com três desenvolvedores, que também se dedicavam a outros aspectos do projeto. Em conjunto com outros membros do departamento começamos a levantar os requisitos que levariam a criação dos 3 portais, dialogando com os políticos responsáveis pelo projeto, com pessoas dos Pontos de Cultura que utilizariam os sistemas e também conceituando a ação de Cultura Digital baseado na pesquisa que desempenhávamos. Este período de planejamento se refletiu em outras equipes do departamento e conceituou o que viria a ser chamado de programa de implantação de Cultura Digital nos Pontos de Cultura. Inicialmente equipes de pesquisa deveriam se familiarizar com *softwares* livres para produção multimídia e documentar esta pesquisa, criando metodologias de ensino das ferramentas. As necessidades destas equipes criaram a demanda de desenvolvimento do Estúdio Livre. Equipes de mapeamento iriam visitar os Pontos de Cultura para conhecer melhor sua realidade local, fornecendo informações importantes para futuras ações e também para o trabalho dos pesquisadores. Das necessidades destas equipes surgiu a demanda pelo MapSys. Além disso, todo o trabalho deveria ser feito em um processo de gestão compartilhada constante de consulta a pessoas dos Pontos de Cultura, foi essa necessidade que incentivou a criação do Conversê.

Durante os primeiros 4 meses de trabalho esboços dos portais foram produzidos e alguns *spykes* colocados em produção para avaliação de usuários dos pontos de cultura e da equipe. Este desenvolvimento aconteceu de maneira bastante orgânica, contando com um Wiki para organização de demandas e idéias. No quinto mês contratamos mais 2 desenvolvedores (com pouca experiência) e resolvemos iniciar um processo de adoção de XP. Durante uma semana juntamos em um mesmo local os desenvolvedores e os pesquisadores que iriam atuar como clientes *proxy* (que estavam distribuídos em outras regiões do país). Durante este tempo conseguimos realizar um jogo do planejamento e, de maneira ágil, colocar em produção a primeira versão de cada sistema. Além disso geramos os documentos exigidos pela burocracia governamental. De todas as práticas de XP, testes e refatorações não foram adotadas devido a inexperiência da equipe, a falta de tempo para aprendizado e a adaptação de softwares livres existentes, cujas comunidades não adotavam métodos ágeis, o que implicou também que um ritmo sustentável não fosse exercido. Nesta semana de iterações diárias que culminaram em 3 *releases* trabalhamos mais do que 10 horas por dia. Mesmo assim, a proximidade dos clientes *proxy* ajudou para que os sistemas produzidos tivessem as funcionalidades mais importantes para serem valiosos em produção (e foram colocados em produção no final da semana) e para que uma metáfora comum fosse construída. A Figura 3.17 mostra um cliente *proxy* pareando com um desenvolvedor

na nossa primeira semana extrema.



Figura 3.17: Cliente *proxy* pareando com um desenvolvedor na Cultura Digital.

A partir deste momento extremo, iniciamos um processo mais cuidadoso de ensino de XP, resolvemos começar a adotar as práticas aos poucos, sendo que eu atuaria como treinador, mas devido a outras responsabilidades (como o processo de licitação dos *kits* que durou mais de um ano) não poderia estar presente o tempo todo. Inicialmente adotamos o jogo do planejamento onde alguns clientes *proxy* criavam histórias e atuavam em conjunto para negociar prioridades entre os três sistemas sendo produzidos. Adotamos também iterações semanais, algumas vezes dedicando uma semana somente a um sistema e depois revezando entre sistemas. Após dois meses, novos *releases* dos 3 sistemas foram colocados em produção e começaram a ser utilizados por mais usuários nos pontos de cultura e pela equipe inteira do departamento que atuava localmente em mapeamentos e oficinas.

Optamos por desenvolver todos os sistemas como adaptações de *softwares* livres existentes: para o Conversê adaptamos o Drupal, para o Estúdio Livre adaptamos o TikiWiki, e para o MapSys resolvemos ser extremos, refatorando ou descartando o código do primeiro *release* e adaptando o TikiWiki para criar o segundo *release*.

Durante estes dois meses adotamos também as práticas de espaço aberto e comunicativo, programação pareada, integração contínua, e papos-em-pé. Tivemos problemas com o espaço aberto, pois a equipe do departamento era multi-disciplinar e algumas vezes os programadores não conseguiam se concentrar em um ambiente altamente caótico, no qual artistas muitas vezes faziam muito barulho (o ápice da crise foi uma gravação de um conjunto de percussão). Além disso, programação pareada não era utilizada 100% do tempo, em algumas tarefas complexas os desenvolvedores com mais experiência trabalhavam sozinhos, em tarefas simples os novos estagiários trabalhavam sozinhos. Era complicado colaborar com membros das equipes distribuídas geograficamente. De qualquer forma, programação pareada se mostrou valiosa para treinar os programadores menos experientes nas plataformas livres que estávamos aprimorando, especialmente a estagiária que entrou para a equipe nos últimos meses. A prática de integração contínua se mostrou fácil de adotar pois seu uso é comum em comunidades de software livre, nas quais nossos desenvolvedores tinham atuado previamente. A prática de papo-em-pé mostrou-se eficiente para melhorar a comunicação entre a equipe.

Ainda neste primeiro ano conseguimos realizar mais um *releases* de cada sistema, sendo que em média utilizávamos 8 iterações de uma semana para produzir um *release* para um sistema. Por mais que os usuários finais dos sistemas estavam satisfeitos com o progresso, os políticos responsáveis pelo financiamento do projeto não conseguiam acompanhar este ritmo de entregas e não se comunicavam com a equipe, reunindo-se poucas vezes com representantes do projeto que tentavam mostrar o valor dos sistemas sendo desenvolvidos. Durante este tempo duas demandas foram feitas por estes políticos, a implementação de logotipos governamentais em todos sistemas e a integração do MapSys com um sistema legado de controle de convênios do Ministério, chamado SGE.

Observamos a extrema importância de ter clientes *proxy* dentro de nossa equipe em contato com usuários finais dos sistemas nos pontos de cultura para guiar o planejamento, foi somente através do seu trabalho que conseguimos balancear as necessidades políticas com as reais necessidades dos usuários. A Figura 3.18 mostra um jogo do planejamento com um cliente *proxy* da Cultura Digital.

A integração com o sistema SGE mostrou-se um exemplo claro da inabilidade do sistema público de atuar com agilidade. Enquanto entre a priorização desta demanda pelo Secretário do departamento



Figura 3.18: Cliente *proxy* durante jogo do planejamento na Cultura Digital.

e a primeira reunião com a equipe de desenvolvimento do sistema legado no Ministério 4 meses se passaram, nossa equipe desenvolveu melhorias nos outros sistemas e priorizou o desenvolvimento do Conversê e o Estúdio Livre. Após esta reunião ficou decidido que iríamos desenvolver um módulo de exportação de dados diretamente no sistema SGE, porém só conseguimos obter uma cópia do código fonte em maio do segundo ano.

Com o código fonte, conseguimos produzir o módulo de exportação em apenas duas semanas, porém os funcionários do Ministério não conseguiram colocá-lo em produção até hoje. Observamos a verdadeira diferença de agilidade de uma equipe que adota XP e de outra que segue processos burocráticos e acaba perdendo velocidade por causa destes.

No início deste segundo ano do projeto encontramos nossas maiores dificuldades. O Secretário

estava insatisfeito com aspectos do projeto e resolveu esperar para renovar o contrato de toda a equipe (que incluía nossa equipe de desenvolvimento). A renovação só aconteceu após o carnaval de 2006, o que causou a saída de 2 dos nossos desenvolvedores.

Na retrospectiva do primeiro ano avaliamos que a adoção de XP andava bem, mas a quebra de continuidade fez com que tivéssemos que começar tudo novamente. Além disso, por atuar em outras demandas, eu não pude fazer meu trabalho de treinador de maneira eficiente, estando ausente muito tempo.

No início do segundo ano tivemos que perder tempo contratando novos desenvolvedores, além disso uma re-estruturação interna no departamento fez com que o nosso espaço aberto em Brasília fosse transformado em cubículos, provavelmente devido a pressão política dos outros funcionários públicos que não se sentiam confortáveis com nossa produtividade e a visibilidade do andamento do nosso projeto. Por isso resolvemos alocar a equipe de desenvolvimento completamente em São Paulo em um novo espaço, desta vez um pouco mais reservado do resto da equipe. Neste ano estruturamos a equipe com 3 novos desenvolvedores (novamente, com pouca experiência) e delegamos o papel de cliente *proxy* a apenas 3 pessoas, onde cada uma representava somente um sistema. A Figura 3.19 mostra o quadro de histórias de algumas iterações da equipe.

Novamente a adoção de XP não foi tão eficiente pois como treinador eu estava muito ausente em outras demandas. Mesmo assim, durante este ano conseguimos treinar os novos desenvolvedores em algumas práticas e estabelecemos um ritmo bem satisfatório de *releases* pequenos. No início do ano conseguimos entregar uma versão de um dos sistemas a cada duas semanas, no final do ano estávamos entregando dois *releases* de sistemas diferentes por semana. Além disso, no final do ano finalmente conseguimos adotar a prática de testes. A Figura 3.20 mostra dois desenvolvedores pareando nesse segundo ano.

Infelizmente, no início do terceiro ano do projeto tivemos problemas políticos na renovação do contrato novamente, devido principalmente a eleição presidencial, perdendo 2 desenvolvedores.

Acreditamos que as maiores dificuldades encontradas foram devidas a ausência freqüente do treinador, a mudança brusca de ritmo e de equipe imposta por fatores políticos, e a inexperiência dos desenvolvedores. O uso de clientes *proxy* foi importante para desenvolver aquilo que os usuários finais necessitavam, porém foi um trabalho árduo para os clientes e causou conflitos na hora de priorizar demandas de diferentes sistemas.



Figura 3.19: Quadro de Histórias de iterações da Cultura Digital.

Além disso, a pouca participação dos políticos que controlavam o orçamento fez com que estes não percebessem as vantagens do processo ágil e não enxergassem valor nos sistemas sendo criados. Na maioria das vezes fomos ágeis demais para os funcionários públicos e tentativas de integração com a equipe do Ministério e sistemas legados, por mais que tentamos algumas vezes, foram sempre frustradas pela inércia do processo burocrático que estes adotavam.

A equipe continua com as práticas de XP e está re-estabelecendo um ritmo de produção aceitável. Após este segundo ano de trabalho a equipe mostrou-se independente do treinador (mesmo com muitas ausências deste no processo de ensino da metodologia) e hoje guia o seu próprio processo, explorando possibilidades de colaboração remota e adaptando o jogo do planejamento para lidar com demandas técnicas de evolução dos sistemas e equilibrá-las com vontades das comunidades de usuários que se formaram em torno de cada portal. Acreditamos que isso foi principalmente devido ao uso de

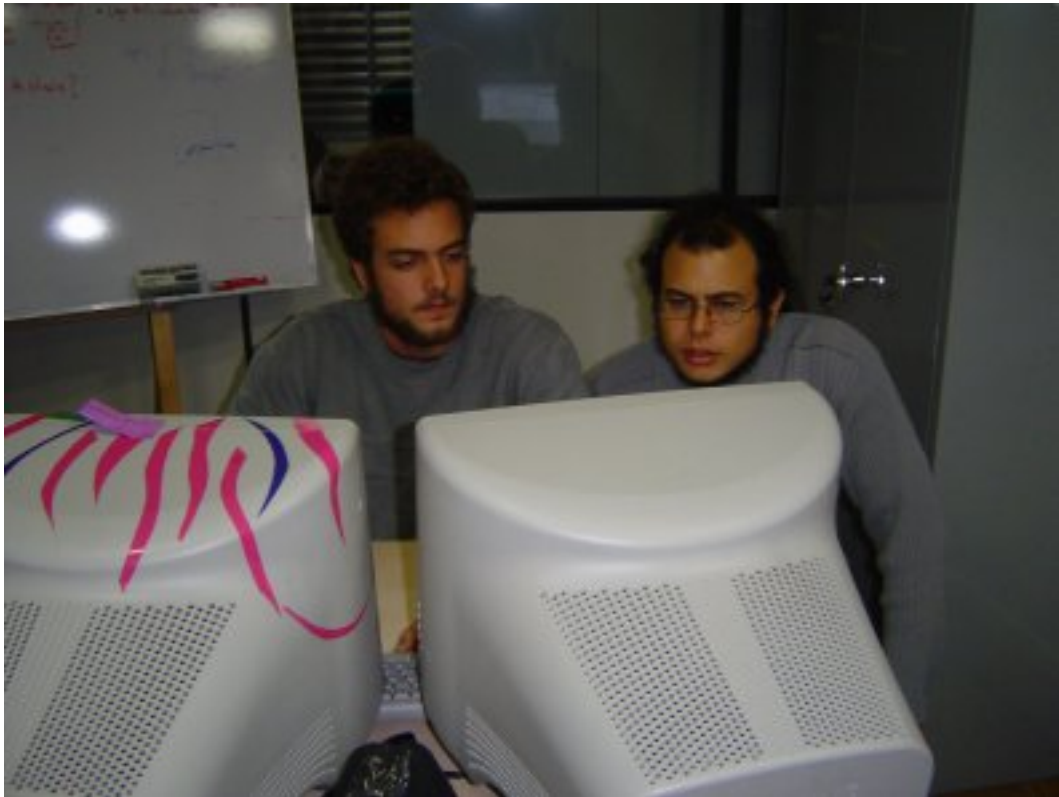


Figura 3.20: Programação pareada no novo espaço da Cultura Digital.

retrospectivas e papos-em-pé que criaram um ambiente de aprendizado contínuo no departamento.

É interessante relatar que algumas práticas ágeis usadas pela equipe de desenvolvimento foram adotadas pelo resto da equipe da Cultura Digital com 80 pessoas espalhadas pelo Brasil, principalmente o jogo do planejamento e retrospectivas.

### **3.7 Workshops, cursos introdutórios, jogos e eventos com a comunidade**

Além dessas experiências em cursos mais longos ou adaptações a empresas ou equipes de funcionários governamentais, *workshops*, cursos introdutórios e eventos como jogos ou conferências são úteis e importantes aliados no ensino de XP. Estas modalidades de ensino são menos eficazes para implantar a metodologia, porém são importantes para divulgar o conceito e também providenciar

### 3.7. WORKSHOPS, CURSOS INTRODUTÓRIOS, JOGOS E EVENTOS COM A COMUNIDADE 95

informação para que pessoas possam optar por adotar a metodologia. Nos relatos que vimos, muitas vezes XP aparece como alternativa após algum membro de uma equipe de desenvolvimento entrar em contato com a metodologia através de palestras ou livros.

Participamos de algumas consultorias rápidas onde em um dia apresentamos, através de palestras, os conceitos da programação extrema. Elas foram importantes para introduzir o tema em empresas e universidades pelo Brasil todo. Foi também a maneira que encontramos de fazer propaganda do método em conferências e festivais da área. Estas palestras normalmente são seguidas por pedidos de cursos médios ou consultorias.

Além disso, já experimentamos cursos curtos, de um ou dois dias com grupos pequenos que participam de uma pequena iteração de XP. Nestes, nosso objetivo é aproveitar poucos dias para mostrar a uma organização como funciona a metodologia e como poderia ser feita uma transição do modo de trabalhar da organização para XP. Normalmente apresentamos rapidamente a teoria e depois simulamos uma iteração de um dia, passando por todas as práticas em um projeto simples e rápido.

Ultimamente temos oferecido dois cursos semanais, um teórico e outro prático, no teórico atendendo também a demanda por informação dos gerentes e administradores de empresa, que podem fazer com que sua equipe técnica participe do prático para aprimorar seus conhecimentos sobre a metodologia. Os cursos foram oferecidos pelo CEC (Centro de Ensino de Computação) do IME-USP tanto para alunos e empresas durante o semestre, quanto como cursos de extensão que acontecem durante os Cursos de Verão do Instituto. O curso teórico conta com exposições sobre diversos temas ligados a metodologias ágeis, aprofundando-se em aspectos mais técnicos como testes e refatoração. O curso prático é uma vivência real onde durante uma semana pequenas equipes participam de um projeto e utilizam todas as práticas de XP com auxílio de um treinador experiente. Nas últimas edições deste curso equipes desenvolveram um web-site para comércio eletrônico de pizzas, o Agil-Pizza. Normalmente fazemos iterações diárias e de 2 a 3 *releases* do sistema durante a semana. Utilizamos duas pessoas como clientes proxy, simulando com bastante fidelidade o relacionamento com clientes do mundo real.

Além desses cursos, já realizamos dois tipos de jogos. A *eXPeriene*, e o *XP Game*, ambos rápidos e divertidos onde algumas equipes de poucas pessoas participam de uma atividade lúdica com o intuito de aprender alguma prática ou entender melhor os valores de XP.

Na *eXPerience* histórias eram criadas para 4 equipes de 4 pessoas envolvendo a construção de



Figura 3.21: Jogadores em um XP Game trabalham para resolver uma história.

estruturas usando peças magnéticas. Cada história já tinha uma estimativa, os jogadores experimentavam o jogo do planejamento dando prioridade as histórias que seriam executadas em 4 iterações curtas no jogo. Os jogadores exercitavam também programação pareada já que poderiam resolver duas histórias por iteração por grupo em duas duplas, porém só uma das pessoas da dupla poderia mexer nas peças.

Já no XP Game a atividade é bem parecida porém todo o grupo trabalha para resolver uma história por vez e o foco é passar conceitos de estimativa e priorização de histórias. Neste jogo as histórias já tem um valor definido pelo cliente, e em cada rodada as equipes devem estimar e priorizar as histórias que vai realizar. Normalmente acontecem 3 rodadas, separadas em 3 fases, na primeira os jogadores estimam as histórias, na segunda dão prioridade às histórias que acham que poderão completar na última fase, de execução. Todas histórias são simples e envolvem dobras com papel,

### 3.7. WORKSHOPS, CURSOS INTRODUTÓRIOS, JOGOS E EVENTOS COM A COMUNIDADE<sup>97</sup>

operações aritméticas simples, organização de cartas de um baralho e até manipulação de bexigas. Ao final do jogo os participantes compreendem também que a velocidade de cada equipe varia de acordo com as estimativas subjetivas que esta aprimora ao longo de iterações. A Figura 3.21 mostra uma equipe trabalhando em uma de suas histórias.

Participar de conferências também é muito importante, em primeiro lugar para trocar idéias e experiências com outros praticantes e estudiosos, em segundo pois é um excelente lugar para conhecer novas técnicas. Nós mesmos aprendemos sobre algumas novas práticas ao participar de conferências, conhecemos alguns jogos, retrospectivas e também a técnica de Mapas Mentais<sup>4</sup>.

Devemos ressaltar o papel da comunidade no ensino de XP, sem seu apoio novidades não chegam aos implementadores com rapidez e não se compartilhariam as experiências que contribuem para o amadurecimento da metodologia. Em especial, a nossa iniciativa de criar a AgilCoop, uma cooperativa de alunos e professores do IME/USP que se especializam no estudo de métodos ágeis, vem sendo elogiada. Uma das atividades que julgamos ser mais importantes na divulgação de métodos ágeis é disponibilizar *podcasts* sobre assuntos ligados a XP e práticas ágeis, possibilitando que muitas pessoas entrem em contato com essas idéias precisando somente entrar no nosso portal e ouvir nossos programas. Acreditamos que investir na divulgação da metodologia é investir na sua aceitação no mercado, gerando assim mais oportunidades de atuação com objetivo de melhorar a qualidade de produção de software no Brasil.

---

<sup>4</sup>Um tipo de diagrama voltado para a gestão de informações, de conhecimento e de capital intelectual; para a compreensão e solução de problemas; na memorização e aprendizado; na criação de manuais, livros e palestras; como ferramenta de brainstorming; e no auxílio da gestão estratégica de uma empresa ou negócio.



## Capítulo 4

# Análise e reflexão sobre o ensino de XP

Os vários casos estudados mostram que XP pode ser aplicada em contextos variados e que o sucesso não depende somente do modo como se ensina a metodologia. Relatos mostram experiências efetivas desde projetos que começaram após um membro de uma equipe estudar XP e conhecer o método teoricamente, até projetos longos com acompanhamento de treinadores experientes, equipe com experiência prévia na metodologia e apoio institucional.

Acreditamos que o objetivo final do educador deve ser formar uma equipe proficiente na metodologia que, após algum tempo em que recebe apoio de um treinador, estará apta a controlar os seus próprios processos, refinando suas práticas em um ambiente de aprendizado contínuo no qual a própria metodologia evolui na realidade local. Para alcançar esta meta as diversas abordagens de ensino de XP se complementam, pois a transição de uma organização normalmente ocorre em várias fases. Como Beck sugere em seu segundo livro [19], a melhor maneira de implantar XP em uma organização é de forma incremental, em pequenos passos, adaptando a teoria ao contexto local. Linda Rising e Mary Lynn Manns sugerem, em seu livro sobre padrões para introduzir novas idéias, que pessoas e organizações passam por três etapas até adotarem completamente uma nova idéia: uma etapa inicial de convencimento, uma etapa durante o processo de implantação da idéia e a etapa de amadurecimento após a idéia estar em uso a algum tempo. Existe também uma etapa, paralela as outras, de resistência [124].

Neste capítulo iremos analisar as experiências de ensino relatadas e refletir sobre maneiras de tornar a transição para a metodologia mais efetiva em diversos contextos, apresentando linguagens de padrões e anti-padrões que reconhecemos nessas diferentes etapas do ensino de XP. Estas linguagens descrevem práticas e processos, além de alguns erros comuns, e podem ser usadas por uma organização

que pretende capacitar seus membros a adotar uma metodologia ágil como XP.

Antes, vale ressaltar uma característica comum a todos os casos que estudamos presencialmente: o fato que eles se deram no Brasil. Peculiaridades do contexto e cultura brasileiros podem influenciar o ensino de XP. De acordo com a teoria do *homem cordial* de Sérgio Buarque de Holanda brasileiros tendem a reagir com seu coração, desenvolvendo uma necessidade de estabelecer contatos amigáveis e encurtar distâncias. Brasileiros rejeitam o uso de sobrenomes e formalidades, referindo-se a todos pelos seus apelidos, até mesmo no espaço de trabalho. Somos incapazes de seguir hierarquias, ou obedecer disciplinas, muito rígidas [40]. Relativo ao ensino de XP existe um impacto positivo: somos abertos a novas experiências, criativos, amigáveis e colaboradores, porém, o impacto negativo também existe: comparados à maioria das culturas do hemisfério norte tendemos a não ser pontuais e a extrapolar prazos. Beck comenta que XP é altamente influenciado pela cultura norte-americana e talvez não funcione bem em uma cultura como a nossa, chegando a mencionar no prefácio ao livro de Vinicius que o maior empecilho à XP no país talvez seja a falta de comprometimento com prazos (mesmo quando eles podem ser ultrapassados por que toda equipe está se divertindo no trabalho) [146].

Acreditamos que é importante levar estas questões em consideração na hora de analisar experiências externas e tentar “tropicalizar” o ensino de XP para que ele se adapte melhor a nossa cultura. Muitos dos padrões que iremos apresentar são reconhecidos na literatura [16, 19, 7, 8, 26, 31, 124, 36, 56, 25, 84], escolhemos detalhar aqueles que em nossa experiência se mostraram valiosos para facilitar o processo de ensino e que se adaptam ao contexto brasileiro. Iremos traduzir e “remixar” padrões já documentados, que representam práticas da Programação eXtrema e outras metodologias ágeis, mas vamos também listar novas idéias que reconhecemos nas nossas vivências, nas experiências estudadas e em conversas com membros da comunidade. Esperamos que estas idéias poderão auxiliar pessoas assumindo o papel de treinadores XP no futuro.

## 4.1 O cenário ideal

### SERÁ QUE VALE?

Observamos algumas características comuns aos casos de sucesso de adoção de XP. Iremos descrever o que consideramos o cenário ideal para ensinar XP nos três contextos estudados, se o leitor se encontra em situações similares a estas acreditamos fortemente que a sua iniciativa será bem sucedida.

Na academia a etapa inicial requer que um professor ou um grupo de pesquisa se interesse por métodos ágeis e estude-os a fundo. Experimentar a metodologia na prática, ou associar-se a praticantes com experiência no mercado, é essencial. Estruturar uma disciplina é o próximo passo, antes disso oferecer atividades como palestras, jogos e cursos rápidos para divulgar a metodologia para alunos e outros pesquisadores é importante. Acreditamos que uma disciplina optativa nos moldes do laboratório de XP é uma boa alternativa, porém XP também pode ser ensinada com eficácia como uma opção alternativa na disciplina de Engenharia de Software, ou ainda em disciplinas de projetos de final do curso de Ciência da Computação. A disciplina deve ter duração entre um semestre e um ano, sendo que poucas aulas são utilizadas no início para apresentar a teoria e o maior tempo dos alunos deve ser dedicado a produção de software seguindo a metodologia, em pelo menos duas aulas de 3 horas semanais. Em uma das aulas comida é oferecida aos participantes. No cenário ideal alunos que já cursaram a disciplina se especializam como treinadores de um grupo de até 12 pessoas. O treinador também atua como mentor dos alunos, se responsabilizando por ensinar técnicas mais complexas como testes e refatorações, ele também é um dos desenvolvedores, pareando durante as aulas. Os alunos que participam da metodologia tem os conhecimentos técnicos básicos necessários (experiência com orientação a objetos, sistemas de *build*, integração contínua e padrões de *design*) e já passaram por experiências de desenvolvimento no mercado. Um dos alunos se especializa no papel de acompanhador. Este grupo deve ter acesso a um laboratório dedicado, configurado como um espaço aberto e comunicativo, com toda infra-estrutura necessária (quadros brancos, computadores para parear, repositórios de código, wikis, etc...) funcionando e com um administrador dedicado a sua manutenção. Os grupos trabalham continuando o desenvolvimento de projetos de software livre utilizados por organizações parceiras da Universidade, existem algumas opções, que são interessantes tanto pela aplicação em si, quanto pela tecnologia que utilizam. Estes sistemas já devem estar em produção, suas versões anteriores foram desenvolvidas usando XP, contam com testes automatizados e clientes *proxy* estão disponíveis para trabalhar com as equipes durante as aulas.

Na indústria a etapa inicial é diferente se a organização que vai adotar XP já foi formada ou não. O segundo caso é mais fácil, se você está criando uma nova organização e pretende que ela utilize XP basta garantir que a equipe inicial seja pequena, composta por ao menos um meta-treinador experiente, dois ou três desenvolvedores habilidosos, um ou dois estagiários e um cliente. Garanta que esta equipe tenha tempo para aprender as técnicas necessárias e que ela conte com um espaço aberto com toda a infra-estrutura necessária. Quando surgir a necessidade de ampliar é possível que os desenvolvedores mais experientes e habilidosos assumam o papel de treinadores, novas contratações

devem levar em conta o ambiente ágil e se adaptar rapidamente ao modo de trabalho da equipe existente, pessoas que não se adaptarem se retiram. O software é produzido utilizando ferramentas e arcabouços livres. Já no caso de uma organização existente, é importante começar convencendo o nível administrativo, organizando palestras, jogos, reuniões com treinadores experientes e até mesmo fazendo com que membros da equipe participem de cursos rápidos. Escolha um grupo pequeno com desenvolvedores experientes e contrate um meta-treinador para auxiliar este grupo no aprendizado de XP durante um projeto piloto no qual um cliente está disponível.

No governo começa organizando palestras para divulgar a metodologia. Quando sua idéia contar com a força política necessária, organize um curso rápido para alguns funcionários. Contrate um treinador experiente para atuar como meta-treinador desta equipe em um projeto de médio prazo para o qual um cliente *proxy* bom esteja disponível. Organize um espaço próprio para a equipe e adote ferramentas e arcabouços livres.

Sabemos que situações ideais raramente ocorrem, nas próximas seções iremos analisar os casos que estudamos e refletir sobre padrões que podem ser adotados nas diferentes etapas do ensino de XP para se aproximar destas situações ideais.

## 4.2 Etapa inicial - Começando o processo de transição

Observamos que o processo de transição metodológico para XP normalmente começa a partir da iniciativa de um ou poucos indivíduos dentro de uma organização. Em geral, uma pessoa assume a responsabilidade de guiar a adoção da idéia até esta ser colocada em prática. Durante este processo é preciso preencher os papéis de treinador, cliente e estruturar uma equipe de desenvolvimento. Deve-se também definir qual o sistema que será desenvolvido e articular um plano de longo prazo com os temas do projeto. Montar um espaço de trabalho com a devida infra-estrutura é um dos últimos passos. A etapa inicial termina logo após a fase de exploração do primeiro ciclo de estação, quando a equipe entra em um ciclo de iterações de desenvolvimento usando XP como metodologia. Em nosso estudo observamos diferentes maneiras de começar este processo, obviamente os atores envolvidos e seus objetivos variam nos diferentes contextos.

Na academia um professor ou um grupo de pesquisa decide ensinar métodos ágeis aos alunos. Se o objetivo é colocar os alunos em contato com o método, optam por oferecer cursos curtos ou incluir XP como tópico de disciplinas existentes. Já se pretendem capacitar os alunos a trabalharem com a metodologia, propõe alterações a disciplinas existentes (ie.: Engenharia de Software ou Projeto de

Conclusão de Curso) ou a criação de uma nova disciplina optativa (como no laboratório de XP).

Na indústria existem casos de organizações que adotam XP desde sua criação e também transições metodológicas em organizações existentes de diversos tamanhos. Se uma organização está sendo criada, normalmente a idéia surge do nível administrativo ou do responsável pela equipe de desenvolvimento, com objetivo de criar um ambiente de aprendizado contínuo que permite com que a organização cresça. Se a organização já existe, normalmente a idéia é apresentada por um gerente de desenvolvimento ou membro de uma equipe técnica com objetivo de experimentar a metodologia em um projeto específico (ou piloto) para que ela seja validada e aplicada em outras partes da organização.

No governo existem casos de equipes inteiras sub-locadas em alguma instituição para desenvolver projetos específicos, a idéia de adotar XP normalmente parte de membros desta equipe. Em alguns casos gerentes de departamentos de tecnologia da informação resolvem mudar a metodologia de trabalho adotada com objetivo de criar um ambiente de desenvolvimento mais ágil e melhorar o relacionamento com a base de usuários finais dos softwares produzidos.

Tanto na indústria quanto no governo em poucos casos a iniciativa provê diretamente do nível administrativo de uma organização.

#### 4.2.1 Padrões para convencer sua organização a praticar XP

Em qualquer contexto, o primeiro e importante passo nesta empreitada é obter conhecimento teórico sobre os métodos ágeis. Além disso, todos envolvidos devem entender os valores, princípios e práticas de XP. Prepare-se para lidar com resistência de muitas pessoas. Recomendamos antes de tudo artigos e materiais de referência disponíveis na Internet e os livros básicos que referenciamos nesta dissertação. Este padrão se chama **Lição de Casa**.

Durante esta etapa inicial, quanto mais pessoas conhecerem e se interessarem por essas alternativas previamente, maior a chance de sucesso de uma experiência prática. Para isso, distribuir materiais sobre XP para que interessados possam se aprofundar teoricamente também é essencial. Este padrão se chama **Plante as sementes**.

Muitas referências estão disponíveis, principalmente em portais ligados a comunidade ágil onde podemos encontrar artigos, histórias de sucesso, listas de discussão, fóruns e até mesmo *podcasts* e vídeos sobre o tema. Ajude pessoas a conhecerem estes ambientes de interação com a comunidade. Este padrão se chama **e-Forum**.



em eventos foram importantes para organizar cursos rápidos e médios em organizações interessadas por XP.

Divertir-se em jogos como o XP Game e a eXPerience é uma maneira efetiva de entender conceitos filosóficos das metodologias ágeis, além de obter uma idéia geral sobre como funcionam algumas práticas e por que elas poderiam ser úteis em seu contexto. Este padrão se chama **Deixe eles brincarem**.

Cursos rápidos e médios são uma excelente oportunidade de apresentar a metodologia a mais pessoas. Em nossa experiência cursos de um ou dois dias são suficientes para apresentar brevemente a teoria da Programação eXtrema e dar uma idéia mais clara aos participantes de como funciona a metodologia na prática, eles devem contar com uma exposição teórica, exemplos práticos das técnicas mais complexas como refatoração e testes automatizados, além de um simulação rápida das etapas envolvidas em uma iteração XP. Cursos médios, nos moldes dos que oferecemos através da AgilCoop<sup>1</sup> são alternativas tanto para tomadores de decisão, quanto equipes técnicas. Estes padrões se chamam **Curso Rápido e Curso Médio**.

Até mesmo na academia palestras, jogos e cursos são importantes para aguçar o interesse de alunos e despertar o interesse de outros pesquisadores. Na nossa experiência palestras onde alunos do Laboratório de XP apresentam seus projetos foram importantes aliadas de jogos, cursos rápidos e médios para melhorar a receptividade de novos oferecimentos da disciplina.

### **Padrões para lidar com a resistência inicial**

Por mais que nossa cultura seja mais aberta a novas idéias, sempre observamos a etapa de resistência em nossos estudos. Na academia esta resistência é peculiar, pois ao optar por um curso sobre XP alunos já estão convencidos a aprender a nova metodologia, então apresentam muito pouca resistência a idéia. Vale ressaltar que quando encontramos resistência no contexto acadêmico ela vêm de alunos que já tem experiência profissional, em forma de dúvidas sobre aspectos extremos e como implementá-los em um contexto real, onde sabem que irão encontrar mais dificuldades. Observamos ainda que estes alunos são os que realmente conseguem apreciar a mudança proposta pela metodologia, pois sofreram com métodos tradicionais em projetos passados.

---

<sup>1</sup>Um é teórico e apresenta mais detalhes das metodologias ágeis e suas práticas mais complexas, outro é prático e oferece uma primeira vivência em um projeto que funciona seguindo os valores, princípios e práticas das metodologias ágeis.

Em outros contextos encontramos muito mais resistência à mudança, principalmente vinda de administradores e gerentes em empresas, ou burocratas no governo, céticos quanto as práticas “extremas” da metodologia, mas também de desenvolvedores que já tem muita experiência de trabalho programando. Tomadores de decisão questionam práticas que aparentemente utilizam recursos de maneira ineficiente, como programação pareada ou o contrato de escopo negociável, enquanto desenvolvedores criticam a metodologia por temer o desconhecido ou sentir-se ameaçados, como no caso do processo de inspeção contínua instaurado pela prática de propriedade coletiva do código.

Acreditamos que o papel do treinador começa justamente durante esta etapa inicial, pois sua experiência é importante para lidar com esta resistência, debatendo críticas comuns aos métodos ágeis e tranquilizando pessoas com medo de passar pela mudança cultural necessária. O padrão **Mentor** dita que a pessoa que pretende introduzir métodos ágeis em sua organização conte com o apoio de treinadores ou membros conhecidos da comunidade. Recomendamos fortemente que se a pessoa que pretende introduzir métodos ágeis em uma organização não participou de ao menos duas experiências práticas com estes, um mentor seja convidado a participar da transição. Ele pode inclusive ministrar palestras, jogos e cursos para sua organização: quando um agente externo também apóia sua idéia a chance de outras pessoas da sua organização darem valor a ela aumenta. Estes padrões são chamados de **Grande Personalidade** e **Validação Externa**.

Além disso, mentores podem participar de reuniões e almoços com tomadores de decisão para explicar-lhes os benefícios desta nova metodologia ou contar sobre suas experiências de sucesso. Foi em um almoço que conseguimos convencer o CEO de uma empresa que programação pareada era uma boa prática para sua organização usando o argumento do *fator caminhão*<sup>2</sup>. Estes são os padrões **Audiência Real** e **Histórias de Sucesso**.

Quando iniciativas partem de uma equipe em uma organização existente, a resistência normalmente surge de pessoas que ocupam um nível intermediário, gerentes em empresas e burocratas no governo, neste caso cabe ao treinador convencer alguém no nível mais alto da organização de que XP é uma boa idéia. Esta pessoa deve ajudá-lo então a garantir que a transição tenha apoio institucional. Este padrão é chamado de **Patrocinador**.

Outro papel importante que começa durante esta etapa inicial é o papel do cliente. É normalmente em uma negociação com o cliente que decide-se adotar a metodologia. Acreditamos que para lidar

---

<sup>2</sup>O fator caminhão indica quantas pessoas podem ser atropeladas por um caminhão até que uma organização pare de produzir. A programação pareada aumenta o fator caminhão de organizações.

com a resistência inicial de clientes uma alternativa é simular um jogo do planejamento com a presença somente do treinador e do cliente, onde temas de um ciclo de estação são discutidos e histórias são criadas levando em consideração o longo prazo de um projeto. Em nossa experiência uma reunião inicial somente com o cliente na Paggo, na qual realizamos um planejamento assim, foi o suficiente para finalmente convencê-los a contratar um treinador e adotar XP, na experiência analisada por Vinicius [147] este padrão também foi utilizado para convencer clientes a adotar XP. Pode-se escolher um dos temas discutidos neste encontro para estruturar um primeiro projeto onde uma equipe vai aprender XP. Chamamos este padrão de **Primeiro Planejamento VIP**.

#### 4.2.2 Padrões para escolher um treinador e se envolver realmente com o cliente

Vimos que preencher os papéis do treinador e do cliente é importantíssimo para garantir o sucesso em qualquer contexto. Por mais que acreditamos que participantes de um curso médio podem começar uma transição, adotando algumas práticas em um ritmo de pequenos passos, recomendamos fortemente que, se uma organização pretende adotar XP ou outra metodologia ágil, que inclua no seu processo de adoção uma etapa de aprendizado mais longa, com apoio de um treinador experiente. As modalidades de ensino mais curtas não são suficientes para capacitar alguém a praticar a metodologia de maneira autônoma e independente pela primeira vez. Por mais que observamos casos onde uma ou poucas pessoas sem experiência prática prévia implantaram XP, esses casos são os que passaram por mais problemas, chamamos esse anti-padrão de **Faça Você Mesmo**.

Observamos que na indústria e no governo o formato mais recomendado é que a equipe que pretende adotar XP contrate um meta-treinador como consultor para dar o apoio necessário durante o processo de ensino da metodologia, Chamamos este padrão de **Meta-treinador** e ele funcionou muito bem na Paggo. Em nossa experiência no governo, a pessoa que deveria assumir o papel de treinador tinha muitas outras responsabilidades, de maneira que o processo de ensino foi mais tortuoso e longo do que esperado.

Na academia recomendamos que um curso mais longo, com ênfase na prática, também seja oferecido. Observamos através de nossas experiências que alunos que participam deste tipo de curso estão aptos a praticar XP em uma organização. Recomendamos que, se o aluno pretende ser o treinador de uma equipe transitando para XP, este participe de ao menos duas experiências práticas com a metodologia. Observamos que o melhor formato para um curso acadêmico permite que alunos participem de um curso de XP duas vezes, na segunda vez se especializando no papel de treinadores. Chamamos este padrão de **Faça Duas Vezes** e observamos que alunos que tiveram esta experiência

estão aptos a treinar uma equipe.

Contar com um cliente real e garantir sua presença desde o início do aprendizado de XP pode ser decisivo para o sucesso de uma experiência. Na Paggo tivemos sorte de contar com um cliente interno a organização, entusiasta das práticas ágeis de XP mas esta não é a norma. No governo observa-se uma situação peculiar, pois a maioria dos sistemas desenvolvidos atendem a uma grande base de usuários (são serviços públicos afinal) e é difícil definir um cliente, muitas vezes até os próprios desenvolvedores são potenciais clientes. Na academia muitos cursos optam por não definir um cliente real, porém os casos de maior sucesso contam com clientes externos às equipes de desenvolvimento e até mesmo à própria instituição. Para garantir clientes interessantes, elaborar projetos com parceiros da Universidade oferecendo um sistema livre desenvolvido sob medida para suas necessidades em troca de um representante para participar de jogos do planejamento, apoiar a equipe e escrever testes de aceitação, é uma excelente idéia, chamamos esse padrão de **Use Clientes Reais**. Em todos estes contextos vale aplicar o padrão **Cliente Proxy** que explicitamos na Seção 2.6. No caso do curso médio prático da AgilCoop, observamos que utilizar duas pessoas como clientes *proxy* cria um ambiente muito mais próximo do real na simulação de desenvolvimento do portal AgilPizza pois ao interagir com pessoas diferentes mudanças de idéia são garantidas com maior frequência. Na industria também se aplicam os padrões **Contrato de Escopo Negociável** e **Pague pelo Uso** reconhecidos por Beck [19]. Ambos ajudam uma empresa a estabelecer melhor o papel de cliente, ao assinar um contrato de escopo negociável um cliente potencial está se comprometendo a colaborar como a metodologia prevê, ao cobrar pelo uso de um sistema a empresa faz com que todos usuários deste componham o papel do cliente. Algo similar pode acontecer no governo, no projeto da Cultura Digital a comunidade de usuários dos sistemas contava com canais para interagir com os desenvolvedores diretamente, podendo participar de discussões em listas e eleger um cliente *proxy* para participar de jogos do planejamento com a equipe, chamamos este padrão de **Comunidade de usuários ativa**.

Em especial, no contexto acadêmico, observamos um anti-padrão interessante denominado **Personalidades Múltiplas** que descreve o erro comum de uma mesma pessoa assumir ambos os papéis de treinador e cliente, podendo levar a resultados esquizofrênicos [56]. Assumir ambos papéis, reconhecidamente trabalhosos e complexos, é uma tarefa estressante e pode confundir a pessoa e os desenvolvedores da equipe, que não sabem dizer se a pessoa está guiando o time e fazendo com que todos sigam as regras do jogo, como treinador; ou se ela esta tentando escrever, validar histórias, ou prover *feedback* claro sobre requisitos, como cliente. Este problema não é difícil de resolver se existe

coragem e disciplina, ou se a equipe é pequena. Quando não é possível designar outra pessoa para assumir o papel de cliente *proxy* ou treinador, usar um chapéu e brinquedos para claramente distinguir o Treinador do Cliente pode ser uma maneira bem humorada de carregar este fardo. Utilizamos esta solução no projeto Cigarra, observamos que o uso do chapéu ajuda a pessoa a concentrar-se para atuar da maneira correta. Conseguimos até simular conversas entre o cliente e o treinador. A Figura 4.2 mostra o uso de um chapéu por um treinador-cliente de personalidades múltiplas.



Figura 4.2: O anti-padrão **Personalidade Múltipla** pode ser resolvido com um chapéu.

### 4.2.3 Padrões para montar uma equipe

Com mais conhecimento em mente os interessados podem partir para estruturar experiências mais prática dentro da organização. Para isso é preciso preencher alguns papéis além do treinador e cliente. O tamanho da equipe que vai adotar XP e sua composição também são importantes, este padrão é

chamado por Beck de **Time Completo** [19] e também é reconhecido por Coplien [31], observa-se que contratar especialistas nos domínios da aplicação garante com que os conhecimentos necessários estarão a disposição de sua empresa. Nota-se ainda que uma pessoa pode preencher mais de um papel necessário à equipe. Recomendamos que equipes não ultrapassem 12 pessoas, nossa única experiência prática com uma equipe de 12 pessoas ocorreu durante o desenvolvimento do Cigarra e acreditamos que ensinar um time grande é uma tarefa complexa. Se sua equipe de desenvolvimento é maior do que isso, separe uma equipe menor para primeira experiência de aprendizado de XP, veremos que é possível escalar a equipe mais adiante no processo de aprendizado. Além disso, é importante contar com pessoas jovens na equipe, o anti-padrão de compor uma equipe sem jovens é reconhecido por Alexander [2] como **Velhos Para Todos os Lados**.

Em todos os contextos é importante preencher na equipe o papel do acompanhador. Este papel não é muito exigente e envolve habilidades de observação e criatividade na criação de radiadores de informação. Na indústria e no governo observamos que estagiários funcionam bem no papel, é o padrão **Estagiário Acompanhador**. Na academia recomendamos que um ou mais alunos de uma equipe assumam este papel, adotando o padrão de **Papéis Secundários** também observado por Dubinsky [45]. Este padrão também pode ser usado para que outros alunos assumam responsabilidades de outros papéis (como treinador, responsável por testes, etc...). Acreditamos que o padrão também pode ser aplicado nos outros contextos, onde é interessante ter um número ímpar de pessoas (3 a 7) assumindo papéis secundários além do papel primário de desenvolvedor, como nota Coplien [31]. No laboratório de XP também recomendamos que os acompanhadores mantenham informações de acompanhamento on-line para que alunos possam trabalhar em suas histórias e atualizar o andamento do projeto quando fora do laboratório, chamamos esse padrão de **Acompanhamento On-Line**.

É claro que a composição da equipe de desenvolvimento é essencial para o sucesso de um projeto. Nota-se que é impossível praticar XP somente com iniciantes. Uma questão que observamos em todos os contextos e pode afetar o andamento de uma transição metodológica é a necessidade de que programadores da equipe cumpram alguns pré-requisitos. É praticamente impossível ensinar refatoração a alguém que não saiba antes programar orientado a objetos. Tentar adotar XP sem antes cumprir estes pré-requisitos é o anti-padrão que chamamos de **Carro na Frente dos Bois**. Como solução na academia é importante indicar aos alunos quais os conhecimentos necessários como pré-requisito para a disciplina. Enquanto na indústria e no governo é importante contar com pelo menos algumas pessoas habilidosas e experientes na equipe, este é o padrão **Contrate Especialistas**. Também é possível treinar pessoas menos experientes ao mesmo tempo em que se ensina a metodologia é o

padrão **Estagiário Aprendiz**.

Em muitos casos a resistência apresentada por desenvolvedores ao aprendizado de XP é justamente devido ao fato que eles desconhecem as técnicas mais complexas exigidas por XP, nesses casos investir em **Cursos Médios** é importante. Além disso, cursos específicos sobre tópicos como programação orientada a objetos podem ser necessários, este é o padrão **Domine os Pré-Requisitos**. Na indústria e no governo também notamos muita resistência de desenvolvedores sêniores, com muita experiência. Por estarem acostumados a sua própria maneira de trabalhar a mudança cultural necessária pode ser muito difícil para eles, evite muitos destes em sua equipe, chamamos este anti-padrão de **Velhos Dinossauros**. Na academia notamos que resistência surge por parte de alunos que já tiveram experiência no mercado, porém recomendamos fortemente que estes alunos participem de disciplinas como o Laboratório de XP, já que vimos que por já terem sofrido com outros métodos na prática, eles dão mais valor as mudanças propostas pela metodologia. Além disso podem apoiar o treinador a engajar a equipe de desenvolvimento. Chamamos este padrão de **Experiência Prévia**.

#### 4.2.4 Padrões para estruturar o espaço de trabalho

O espaço de trabalho também influi muito em uma experiência de aprendizado de XP, a prática de **Espaço de Trabalho Informativo** é um padrão fácil de ser aplicado, recomendamos que o espaço seja amplo, contenha computadores de forma que duas pessoas possam praticar programação pareada, mas que também conte com um espaço privativo para que membros da equipe possam lidar com questões privadas. Se possível estruture a primeira experiência de aprendizado de forma que toda a equipe possa seguir o padrão **Sentar Juntos**, mais adiante veremos como lidar com casos onde a equipe está distribuída geograficamente.

O uso de uma parede e um quadro branco para aplicar o padrão **Radiadores de Informação** também é recomendado. Vimos que o importante destes radiadores é comunicar facilmente a métrica observada, não precisando ser objetos estéticos. Além do espaço de trabalho físico, em nossa experiência na academia incentivamos acompanhadores a estruturar um espaço virtual, disponível on-line, onde métricas e informações são armazenadas. Recomendamos o uso do XPlanner para armazenar histórias e tarefas bem como o andamento do projeto e o uso de um Wiki para que a equipe possa documentar suas idéias e combinados. Chamamos este padrão de **Espaço Virtual Informativo**. No contexto governamental um espaço como estes é recomendado para que a equipe de desenvolvimento possa prestar contas de maneira visível a toda comunidade de usuários dos sistemas desenvolvidos. Equipes distribuídas também podem se beneficiar da adoção deste padrão.

Além do espaço é importante prestar atenção a questões de infra-estrutura básica para que o processo de aprendizagem aconteça sem empecilhos. Estruturar um servidor para que a equipe possa contar com um repositório de código fonte desde o início do projeto é importante. Além disso, recomendamos que a infra-estrutura conte com contas de usuários configuradas com todo um ambiente de desenvolvimento, no qual devem estar instalados os arcabouços, servidores de aplicação, bancos de dados, ferramentas e a documentação da tecnologia que será utilizada no projeto. Recomendamos fortemente que a equipe preencha um papel de administrador que zele pelo funcionamento desta infra-estrutura, é o padrão **Tenha um Root**. Em nossa experiência na academia foi importante contar com alguns alunos bolsistas encarregados de manter a infra-estrutura do laboratório de XP e mesmo assim tivemos problemas com hardware quebrado e serviços instáveis. Na indústria e no governo pode-se contratar uma pessoa ou um grupo para cuidar da administração da rede utilizada pela equipe de desenvolvimento. Em nossa experiência na Cultura Digital, no primeiro ano os desenvolvedores se revezavam no papel de administrador, sendo que cada um escolhia dias nos quais teria uma história reservada no planejamento só para cuidar de questões ligadas à infra-estrutura. O problema é que estes ficavam sobrecarregados quando pessoas de outras equipes precisavam de apoio. No segundo ano contratamos uma pessoa para cuidar somente da infra-estrutura e observamos que a equipe de desenvolvimento pode investir tempo na criação de pequenos programas que melhoravam a sua infra-estrutura de desenvolvimento colaborativo. Em qualquer contexto recomendamos fortemente o uso de softwares livres tanto para arcabouços, quanto servidores de aplicação e ferramentas. A qualidade das soluções existentes já foi comprovada pela comunidade, além do fato de que o código disponível é fonte de aprendizado que pode ser explorada pela equipe de desenvolvimento. Como sistema operacional recomendamos qualquer “sabor” de GNU/Linux, para serviços de repositório de código fonte recomendamos o SubVersion, para ambiente de desenvolvimento recomendamos o Eclipse, que como vimos funciona até como um terceiro ajudante durante programação pareada, além de contar com excelentes ferramentas para refatoração automatizada. Este padrão é chamado de **Use Software Livre**.

Uma parte importante da infra-estrutura, que varia conforme as tecnologias adotadas e a estruturação dos servidores usados pela equipe, é o script de *build*. Dependendo do caso, ele pode ser responsável por compilar a aplicação, rodar testes automatizados, atualizar automaticamente um ambiente de testes e até mesmo colocar *releases* curtos devidamente identificados no repositório em um ambiente de produção. Criar este script o quanto antes e com ajuda de um treinador é recomendado, observamos que uma vez que o script está feito até mesmo desenvolvedores com pouca

experiência podem se beneficiar do seu uso, este padrão é chamado de **Script de Build**.

Se pensarmos o código fonte como um espaço freqüentado diariamente pela equipe de desenvolvimento, compreendemos a importância de estabelecer um **Padrão de Codificação** o quanto antes para estabelecer toda infra-estrutura base necessária para começar um projeto. O Eclipse pode ser usado como um bom aliado para garantir o padrão de codificação adotado, pois conta com um *plug-in* que ajuda a automaticamente gerá-lo, compartilhá-lo e mantê-lo. Na Paggo o padrão de codificação era discutido em reuniões, na maior parte dos casos sugerido por desenvolvedores com mais experiência e colocados em um radiador de informação chamado “Combinados da Equipe”. Foi simples ensinar o padrão e praticá-lo durante sessões de programação pareada. Acreditamos que chamar o padrão de “combinado” e ser flexíveis quanto a sua adoção fez com que ele fosse mais fácil de absorver por membros da equipe menos experientes. É uma maneira de adotar o padrão ao contexto brasileiro, já que somos mais suscetíveis a seguir um combinado do que uma regra.

#### 4.2.5 Padrões para o planejamento inicial

O próximo passo é decidir aplicar XP em um projeto. Assim que soubermos qual será este projeto entramos na etapa do aprendizado durante a qual as pessoas estão amadurecendo a sua prática da metodologia. Recomendamos que este projeto inicial, aonde acontecerá a etapa mais importante do aprendizado, tenha duração média. Em contextos onde os desenvolvedores estão presentes todos os dias acreditamos que 3 meses são o suficiente para aprender XP. Porém, tanto no caso da Paggo quanto na Cultura Digital o aprendizado de XP aconteceu durante mais de um projeto e durou 6 meses e um ano respectivamente. No caso da Paggo, por razões financeiras, o meta-treinador não estava presente todos os dias, podendo acompanhar a equipe somente de dois a três dias por semana. Na Cultura Digital o meta-treinador tinha outras responsabilidades e estava muito ausente, observamos que a equipe demorou mais tempo para amadurecer (as trocas constantes de metade dos desenvolvedores da equipe também não ajudaram).

Já na academia, dependendo da carga horária, disciplinas podem ter duração de 6 meses (para alunos interessados em aprender a usar XP) a um ano (para aqueles que pretendem se especializar no papel de treinadores). Recomendamos o formato de duas aulas semanais, de no mínimo 3 horas. É importante incentivar a presença dos alunos, o padrão **Almoço eXtremo** é uma boa tática para se adotar, além disso os alunos devem investir mais horas semanais no desenvolvimento (e para isso é importante ter acesso ao lab).

Em qualquer contexto a carga de trabalho deve levar em conta o padrão **Trabalho Energizado**. Acreditamos que uma boa carga didática na academia é de 8 horas semanais, no nosso caso alunos conseguem manter suas responsabilidades em outras disciplinas e se dedicar ao projeto sendo desenvolvido no laboratório. Em empresas no Brasil, devido ao contexto econômico, é normal que desenvolvedores estejam dispostos a trabalhar horas extras, em muitos casos sem remuneração. Na Paggo observamos que em épocas estressantes desenvolvedores estavam dispostos a trabalhar horas extras, o fato que estagiários não estavam presentes em tempo integral encorajava-os a colocar horas extras. Recomendamos que a prática de horas extras não torne-se a norma, comprometendo o trabalho energizado.

Em empresas e no governo é interessante que o projeto inicial seja fruto de aplicar o padrão **Planejamento VIP**. Neste caso recomendamos que o tema planejado enfoque em desenvolver um projeto piloto interno durante o período de aprendizagem da metodologia, este é o padrão que chamamos de **Projeto Piloto**. Em alguns casos porém, principalmente com clientes externos, isso não será possível, sendo importante que o cliente entenda as implicações de um contrato de escopo negociável. A equipe XP garante a entrega das funcionalidades mais importantes para o cliente nos primeiros *releases*, porém durante o aprendizado é altamente provável que algumas histórias tenham que ser descartadas e *releases* re-planejados devido a erros de estimativa.

Na academia é importante que os alunos estejam engajados em projetos realísticos. Recomendamos que projetos reais sejam desenvolvidos com parceiros da Universidade e distribuídos como software livre. Diferentes grupos de alunos podem continuar o desenvolvimento destes projetos em instâncias diversas da disciplina. Em cada instância, clientes planejam um tema que possa ser desenvolvido por completo durante um semestre.

O padrão **Ciclo de Estação** define como um primeiro jogo do planejamento pode acontecer com clientes para que um tema seja selecionado e desenvolvido durante a próxima etapa, de amadurecimento do aprendizado de XP. Nesta etapa inicial é importante que o planejamento leve em conta a inexperiência da equipe e adote o padrão **Folga** para que deslizos possam ser recuperados. O padrão **Fase de Exploração** é uma boa oportunidade para que nesta etapa a equipe de desenvolvimento possa entender as histórias do tema explorado. Vale mencionar que **Histórias** é um padrão de como capturar requisitos de um sistema. O padrão **Spike** permite com que desenvolvedores explorem as possibilidades tecnológicas de domínio da aplicação a ser desenvolvida. Recomendamos que o treinador participe de programação pareada com desenvolvedores e ajude-os a criar spikes, podendo introduzir tópicos como testes e refatoração em código que poderá ser descartado. Além disso, cons-

truir o seu sistema utilizando arcabouços livres como base (como XUnit, Ruby on Rails, Hibernate2, VRaptor e outros) é uma excelente idéia. Muitos desses arcabouços foram criados levando desenvolvimento ágil em consideração e a quantidade de código que pode ser reutilizado é enorme, chamamos este padrão de **Use Arcabouços Livres**. A maioria dos arcabouços recomendados contam inclusive com código de testes que pode servir de exemplo para o aprendizado.

### Bootstrap

Um anti-padrão interessante que observamos nesta última fase da etapa inicial de aprendizado de XP chamado *Bootstrap* [56] descreve como equipes enfrentam dificuldades para começar um projeto sem uma base de código. Desenvolvedores em equipes maiores tem problemas para integrar partes de um sistema ou até escrever testes quando não existe uma base de código sobre a qual se pode construir. É difícil para pares se revezarem pois o par que começou a codificar um componente tem resistência de deixar outros pares trabalharem nele com medo do que vão pensar do seu código ou dos seus testes, ou até porque ele quer continuar o desenvolvimento do código que ele começou a construir. Coordenar tarefas é complicado, especialmente para a história de *bootstrap* (a primeira história que uma equipe implementa [10]), conseguir que uma equipe trabalhe em paralelo é complicado já que muitas histórias dependem de outras que não foram desenvolvidas.

Notamos que o problema é fácil de resolver, recomendamos quatro alternativas. A primeira é dividir histórias grandes e a história de *bootstrap* usando **Historiótipos** [105], uma linguagem de padrões que define diferentes tipos de histórias<sup>3</sup>. Pode-se também construir a aplicação sobre uma base de código de software livre existente, é o que fazemos atualmente no laboratório de programação extrema, onde projetos como a Colméia, a Borboleta e o Archimedes já entram no seu terceiro ano. Nestes casos observamos que a base de código com testes e *design* simples funciona como uma excelente ferramenta de aprendizado. Outra opção é desenvolver o código de *bootstrap* com uma pequena equipe, adotamos esta estratégia nos cursos médios da AgilCoop, nos quais alunos contam com uma base de código inicial desenvolvida pelos treinadores. Em outros contextos neste caso pode-se fazer uma reunião com a equipe onde o *design* e a metáfora de uma primeira arquitetura é discutido e depois permitir que um par ou sub-grupo de desenvolvedores mais experientes codifiquem as classes base do modelo de negócios rapidamente. Também é possível que a prática de testes seja

---

<sup>3</sup>Historiótipos definem quatro tipos diferentes de histórias concretas, que (1) ajudam a montar a infra-estrutura (montar o servidor de integração contínua, configurar o ambiente de desenvolvimento, aprimorar o script de *build* ou até mesmo pesquisar tecnologias novas), (2) criam nova funcionalidade no sistema, (3) alteram funcionalidade existente e (4) criam novas regras de negócio.

introduzida somente após existir uma base de código, de maneira com que desenvolvedores vão sentir menos medo de contribuir o seu código inicial.

### 4.3 Aprendizado através da repetição prática - Amadurecimento da metodologia

Após a etapa inicial de aprendizado entramos em um ciclo de iterações, no qual a equipe vai amadurecer seu conhecimento da metodologia através da repetição das práticas. Esta etapa normalmente termina quando a equipe tem autonomia para conduzir seu próprio processo independente de um treinador.

Diferentes estratégias podem ser utilizadas, observamos que é eficiente adotar todas práticas primárias desde o princípio quando a equipe é relativamente experiente, na academia acreditamos ser esta a melhor opção. Podemos também adotar práticas aos poucos, no caso de equipes menos experientes essa estratégia é recomendada, é o padrão **Passos Pequenos**. Neste caso é importante levar em conta a dependência que existe entre algumas práticas, a experiência de um bom treinador é valiosa na hora de decidir ou não começar a usar uma prática.

Seja qual for a estratégia adotada é muito importante que o treinador crie um ambiente no qual a equipe de desenvolvimento participe ativamente da construção metodológica, podendo tirar suas dúvidas e expor suas críticas à metodologia, chegando a até mesmo propor alterações ao modo como as coisas estão sendo feitas. Toda informação relevante ao projeto deve chegar a equipe de desenvolvimento, valorizar a comunicação é valorizar a opinião das pessoas mais importantes no processo de desenvolvimento de software: os desenvolvedores. Beck [16] comenta que o desenvolvedor é a figura central da metodologia, Coplien [31] reconhece este padrão como **Desenvolvedores Controlam o Processo**. Acreditamos que uma equipe de desenvolvimento realmente engajada na mudança metodológica é chave para o sucesso.

Em nossa experiência na Cultura Digital observamos que o processo de adoção de XP ocorreu muito lentamente, a prática de testes por exemplo só foi implantada um ano e meio após a equipe começar sua transição metodológica. Chamamos este anti-padrão de **Passos Pequenos Demais** e acreditamos que ele pode ser resolvido com a presença mais constante de um treinador, que deve garantir que o ritmo de evolução das práticas se mantenha razoável e com o padrão **Desenvolvedores Controlam o Processo**. Na nossa experiência a velocidade de mudança na Cultura Digital acelerou consideravelmente quando em uma reunião o treinador e os desenvolvedores chegaram a um acordo de que os próprios desenvolvedores deveriam propor alterações e melhorias ao seu processo, ao invés

### 4.3. APRENDIZADO ATRAVÉS DA REPETIÇÃO PRÁTICA - AMADURECIMENTO DA METODOLOGIA 117

de sempre esperar que o treinador ou o líder da equipe propusesse a maneira como eles deveriam trabalhar.

Já vimos que o padrão **Retrospectivas** encoraja a discussão sobre o processo adotado pela equipe. Para que um ambiente de melhoria e aprendizado contínuo seja a norma da equipe, retrospectivas providenciam um momento de reflexão necessário. Recomendamos fortemente que durante esta etapa do aprendizado retrospectivas sejam realizadas com frequência. Segundo Cockburn [29], este padrão é fundamental para ajudar a equipe a encontrar o processo que melhor se adapta ao seu contexto local.

Um anti-padrão interessante relacionado a esta etapa do aprendizado se chama **Formato Formal** [84]. Ele ocorre quando uma equipe adota e fixa as práticas de XP exatamente como indicado na teoria, o problema é que esquecem de adaptá-las para funcionar melhor em seu contexto e não criam um ambiente de melhoria contínua. A solução é compreender que XP pode ser uma metodologia heurística além de um simples método a ser adotado. A prática de retrospectivas é importante para criar um ambiente de aprendizado e melhoria contínua.

#### 4.3.1 Padrões de treinamento

Vimos que ambos os papéis de treinador e cliente são complexos e exigem dedicação. Alguns padrões observados em nossos estudos podem ser úteis para auxiliar as pessoas que preenchem estes papéis durante a etapa de amadurecimento no aprendizado de XP.

Em qualquer contexto, em especial durante o começo de projetos de aprendizado de XP, recomendamos que o treinador adote o padrão **Treinador também programa**. No início do projeto, ainda mais com desenvolvedores pouco experientes e sem conhecimentos de desenho de arquiteturas orientadas a objetos, é importante que o Treinador participe de sessões de programação pareada e ajude os desenvolvedores a implementar o sistema. Em nossa experiência na Paggo foi valioso tanto parear com desenvolvedores mais experientes para avançar em questões mais complexas, quanto parear com estagiários para ensinar conceitos básicos. No laboratório de XP, em especial no projeto Cigarra, parear com os alunos era importante para mantê-los concentrados nas tarefas que estavam desenvolvendo. Neste contexto observamos que, talvez pelo período de tempo que passa entre uma aula e outra ou até mesmo a inexperiência de alguns com projetos reais, alunos tendem a divagar em suas tarefas e perder a visão global do projeto. O treinador deve sempre estar atento a alunos que perderam o rumo, e ajudá-los a retomar o enfoque no desenvolvimento.

O papel do treinador quando atuando como mentor durante de sessões de programação pareada porém não é resolver todos os problemas e sim incentivar que os próprios desenvolvedores reconheçam soluções, oportunidades para simplificar o sistema e possíveis problemas para testar. Este padrão se chama **Mostre o Caminho**. No caso de equipes com pouca experiência em arquitetura de sistemas orientados a objetos, muitas vezes desenvolvedores implementam soluções que não seguem padrões de *design*. Ao invés de explicitar um padrão que pode ser aplicado na arquitetura, recomendamos que treinadores ensinem alguns padrões para a equipe, escrevam testes para evidenciar problemas e deixem com que a própria equipe reconheça oportunidades de aplicar um padrão em seu tempo. O padrão **O Suficiente** recomenda que, para facilitar o aprendizado de um conceito difícil, treinadores dêem apenas uma breve introdução e disponibilizem outras informações para quando as pessoas estiverem prontas.

Um padrão que foi muito valioso em nossa experiência foi **Diários de um Treinador**. Este padrão sugere que o treinador mantenha um diário no qual pode anotar situações observadas e refletir sobre intervenções necessárias ou questões que precisa discutir com membros da equipe. Tanto no laboratório de XP quanto na Paggo mantivemos um diário onde anotávamos nossas reflexões após um dia de trabalho. Rer ler estas anotações mostrou-se valioso para direcionar nossas intervenções.

Para motivar a equipe o padrão **Pequenos Sucessos** sugere que o treinador comemore até mesmo os pequenos avanços realizados pela equipe. Realizar pequenas celebrações pode melhorar muito a moral dos desenvolvedores. Na Paggo levamos um bolo para comemorar o dia em que conseguimos integrar nossos primeiros testes. Também realizávamos um **Happy Hour** após *releases*, o ambiente informal deste padrão é uma excelente oportunidade para que o treinador converse com membros da equipe individualmente sobre suas percepções e opiniões quanto à transição pela qual estão passando.

No laboratório de XP o dia no qual as equipes apresentam seus *releases* sempre conta com um clima festivo e inclui o **Almoço eXtremo**. O almoço é um bom padrão para incentivar alunos a comparecerem às aulas. Por mais que alguns alunos sejam muito dedicados, em todos os anos observamos outros que faltam muito, especialmente em equipes maiores. **Inclua Comida** é na verdade um padrão, envolver refeições em eventos, cursos e reuniões é uma maneira de atrair participantes e providenciar uma oportunidade de interação com as equipes passando pela experiência de aprendizado em um ambiente mais descontraído. Na Paggo um café da manhã mensal incentiva a equipe a chegar mais cedo e é usado como uma oportunidade de todos se envolverem melhor com o domínio de negócios da empresa, contando com apresentação de notícias ou discussões sobre tendências de mercado diretamente com clientes e *proxys*.

### 4.3. APRENDIZADO ATRAVÉS DA REPETIÇÃO PRÁTICA - AMADURECIMENTO DA METODOLOGIA 119

Valorizar o *feedback* veloz é muito importante para que a equipe possa ser realmente ágil, até mesmo no seu aprendizado de XP. Métricas auxiliam 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. Recomendamos que a pessoa assumindo o papel de acompanhador tenha apoio constante para que métricas evidenciem aquilo que a equipe precisa perceber. A tese de mestrado de Danilo Sato [132] demonstra como fazer uso eficaz de métricas em projetos que utilizam métodos ágeis e conta com um catálogo de métricas que podem ser utilizadas para evidenciar deficiências nas diferentes práticas de XP. Este é o padrão **Auxilie o Acompanhador**.

**Apoiar o Cliente**, ou *proxys* também é uma tarefa importante para o treinador. **Envolvimento Real com o Cliente** envolve a participação ativa deste papel em jogos do planejamento, papos-empé e avaliação de *releases*. Para isso, o padrão **Seja Paciente** sugere que o cliente espere re-planejar e re-priorizar histórias das primeiras iterações de uma equipe que está aprendendo XP, já que a probabilidade de erros de estimativas e defeitos surgirem é mais alta neste período.

Nos contextos industriais e governamentais quando o treinador é contratado como um consultor externo o padrão **Resistência ao Meta-Treinador** indica que é natural que a equipe questione a importância deste ator, já que percebem o investimento em uma pessoa externa como evidência de falhas internas. Neste caso é importante deixar claro que o papel de meta-treinador é temporário e que o quanto antes a equipe se apropriar da metodologia, o meta-treinador poderá completar o seu trabalho. Acreditamos que este argumento incentiva a equipe a dominar XP. Também recomendamos que meta-treinadores diminuam a frequência de sua participação ao longo do tempo.

Em contextos acadêmicos alguns padrões específicos podem auxiliar o treinamento, como **Lista de Presença** que sugere que alunos tenham que assinar uma lista quando comparecem as aulas. Ao levar a participação em conta na hora de avaliar o aluno, incentivamos sua presença nas aulas. Observamos que mesmo assim muitos alunos faltam durante o início do semestre e querem repor suas faltas no final do mesmo. Nestes casos sugerimos o padrão **Tempo Extra** que indica que estes alunos podem repor créditos relacionados a presença se dedicando a tarefas importantes para finalização dos projetos em tempo extra, fora das aulas fixas do laboratório. Além destes, o uso de uma lista de e-mails e um Wiki auxiliam o treinador a manter um diálogo contínuo com a sua equipe. Observamos no projeto Cigarra que a lista de e-mails permitiu com que jogos do planejamento durassem uma aula e todo o tempo até a próxima aula, no qual histórias e tarefas eram discutidas tele-presencialmente. Este é o padrão **Lista de Discussão**.

### 4.3.2 Padrões de planejamento contínuo

Vimos que o projeto inicial de aprendizado pode ser planejado como um tema de um ciclo de estação. Porém é bem possível que uma equipe aprendendo XP passe por mais de um ciclo de estação. Neste caso é sempre importante contar com a fase de exploração e o apoio de *spykes* para que o planejamento mais detalhado seja realizado com as informações necessárias em mente.

O planejamento detalhado de uma iteração acontece durante o chamado **Ciclo Semanal**. Na verdade uma iteração pode conter alguns destes ciclos, o importante é que eles sejam utilizados para entender como anda o projeto e eventualmente fazer ajustes ou no escopo ou na data de entrega de um *release* de uma iteração. Observamos que o ciclo semanal não dura exatamente uma semana. Principalmente no início do aprendizado pode ser importante fazer com que este ciclo tenha duração maior, de até um mês, de maneira com que a equipe não fique sobrecarregada com o *overhead* de tempo implícito nas reuniões de planejamento e *releases*. Em nossa experiência na Paggo começamos com ciclos de um mês e reduzimos eles para 2 semanas. Na Cultura Digital começamos com ciclos de uma semana, mudamos para ciclos de duas semanas e atualmente empregam-se ciclos semanais. Na Academia uma semana é muito pouco se considerarmos a carga didática recomendada para o laboratório de XP, de maneira com que faz mais sentido que este ciclo tenha duração de um mês. Atualmente os ciclos da Paggo duram 2 semanas, porém cada sub-equipe tem ciclos descompassados, os da Cultura Digital duram uma semana, sempre começando na terça-feira.

No planejamento, seja ele o “semanal” ou de estação o padrão **Histórias** recomenda um formato para que requisitos funcionais sejam capturados pela equipe. As **Estimativas** devem ser feitas com apoio de toda equipe de desenvolvimento, é importante porém que exista tempo para que os desenvolvedores se acostumem com o conceito e aprimorem suas estimativas. **Responsabilidade Aceita** implica que o desenvolvedor responsável por uma história aceite esta responsabilidade voluntariamente e concorde com a estimativa anotada. Durante os ciclos semanais o padrão **Velocidade** recomenda que o acompanhador compare as estimativas com o tempo real gasto em tarefas e declare qual a capacidade de produção de uma equipe. É normal que a velocidade de um projeto aumente com o passar do tempo e com a familiarização de uma equipe com estimativas. Principalmente no início de um projeto incluir **Folga** no planejamento é importante para que a equipe não fique desmotivada ao errar estimativas, oferecendo uma oportunidade de reutilizar o tempo de folga em tarefas que demoraram mais do que estimado.

Para auxiliar o planejamento e acompanhamento de um ciclo recomendamos o uso do padrão

**Quadro de Histórias** que sugere que um espaço físico seja separada para que toda equipe (e colaboradores externos) possa visualizar quais histórias estão sendo desenvolvidas, quais foram completadas e quais ainda faltam para terminar um *release*. Este padrão pode ser reconhecido na prática de *backlog* do Scrum [136, 39].

Também para auxiliar na concepção de histórias, principalmente como apoio a um cliente *proxy*, recomendamos aplicar o padrão **Personas** [39]. Personas são pessoas fictícias que representam um grupo de usuários do sistema criadas pela equipe de desenvolvimento para serem atores em histórias. O uso de personas ajuda a equipe a considerar diferentes casos de interação com o sistema sendo desenvolvido, que normalmente são esquecidos quando se pensa em um único tipo de usuário.

Já vimos que é importante que o planejamento sustente o padrão de **Trabalho Energizado**. Para enfatizar esta prática na indústria e no governo recomendamos o uso do padrão **Tempo de Estudo** que pode ser organizado como grupos de estudo continuados. Na Paggo ao ouvir da equipe a demanda por tempo para se aprofundar em novas tecnologias em um primeiro momento incluímos histórias de estudo no planejamento. Cada desenvolvedor poderia se especializar nas tecnologias que lhe interessavam pegando algumas histórias de estudo no planejamento do ciclo semanal [55]. Atualmente a Paggo reserva uma hora por dia para que seus desenvolvedores exercitem ócio criativo e um dia reservado para estudos no seu ciclo (que dura duas semanas). Na Cultura Digital também fizemos uso de histórias de estudo. Tempo de estudo pode evoluir para exploração quando começa a envolver codificação de *spykes*. Na Cultura Digital histórias de estudo eram usadas quando sentíamos que ainda não existia informação suficiente sobre uma tecnologia para colocá-la em prática.

Um bom padrão para observar se o ritmo de trabalho está sustentável é o **Humorômetro**. O humorômetro, ou calendário niko-niko [132] é um cartaz onde membros de equipe anotam o seu humor. Se o calendário estiver marcando muito mal humor pode ser um sinal de que o ritmo não está sustentável.

Na academia acontece de alunos abandonarem a disciplina, isto pode implicar que a equipe desfalcada precise ajustar-se para manter o trabalho energizado. Um anti-padrão neste contexto, chamado **Abandonar o Navio**, é não contar com alunos que cancelam sua matrícula. Recomendamos que equipes só se formem após passado o período de cancelamento. Além disso, se algum aluno precisar abandonar a disciplina é importante que o cliente re-priorize histórias, reduzindo o escopo, que a equipe conte com mais ajuda de seu treinador, ou até mesmo que equipes maiores cedam um integrante para a equipe desfalcada.

No início do aprendizado, quando não existe ainda maturidade na prática de testes, é muito provável que os *releases* pequenos não sejam de altíssima qualidade. Um bom padrão para manter a equipe motivada é negociar com o cliente um **Bug Fix Release**. Na Paggo após uma entrega particularmente defeituosa onde o cliente se recusou a aceitar qualquer uma das histórias, negociou-se uma iteração mais curta (o ciclo duraria uma semana) com o único objetivo de testar melhor a aplicação e eliminar os defeitos observados. Posteriormente descobrimos que um único defeito introduzido na véspera do *release* em uma parte da aplicação fora da cobertura de testes tinha “quebrado” todo o sistema. Consertado o defeito, investimos o resto do tempo para aprimorar os testes. Na Cultura Digital também realizamos alguns *bug fix releases*.

Já discutimos também os padrões **Retrospectiva** e **Papó-Em-Pé**. Recomendamos fortemente seu uso para poder adaptar o planejamento de acordo com o *feedback* constante da equipe. Um efeito colateral ligado a cultura brasileira observado na Paggo foi o uso das ferramentas de retrospectiva para “fofoca” e discussões pessoais. Quando as pessoas esquecem de usar a retrospectiva acreditando que todos na equipe fizeram o seu melhor o anti-padrão **Fofoca** pode se manifestar. Após um dia no qual o desenvolvedor mais sênior saiu mais cedo pois tinha sido avisado que a equipe poderia relaxar antes de um feriado, porém sem comunicar o fato pro resto da equipe, o nosso quadro de histórias na Paggo (também usado para guardar anotações para retrospectiva) transformou-se em uma guerra de reclamações. O padrão **Reunião de Lavar Roupa Suja** mostrou-se importante para solucionar este caso, criando um ambiente onde questões pessoais poderiam ser resolvidas a equipe encontrou uma maneira de acertar suas diferenças [55]. Não recomendamos que este padrão seja utilizado com frequência pois é emocional e intenso.

### 4.3.3 Padrões de *design*

O padrão **Design Incremental** sugere que exista investimento contínuo na arquitetura e desenho de uma aplicação. Acreditamos que adotá-lo é extremamente importante para manter o *design* de um sistema simples de fato. Porém, infelizmente, não podemos sempre contar com a experiência necessária para guiar o desenho de uma arquitetura em uma equipe, especialmente devido a economia Brasileira. Vimos que o treinador deve ajudar neste caso. Acreditamos que aplicar os seguintes padrões é válido em qualquer contexto.

A *Metáfora* é um padrão simples que muitas vezes é negligenciado, sem motivo, no ensino de XP. Para criar uma boa metáfora é importante que o cliente se esforce para ensinar aos desenvolvedores os termos de domínio do negócio e que os desenvolvedores ensinem ao cliente termos relativos às

### 4.3. APRENDIZADO ATRAVÉS DA REPETIÇÃO PRÁTICA - AMADURECIMENTO DA METODOLOGIA123

tecnologias utilizadas. No projeto Mico a metáfora do algoritmo genético foi facilmente assimilada pela equipe e os clientes. Na Paggo reuniões periódicas com o cliente, onde este falava sobre o domínio de negócios da empresa, ajudaram muita na criação de uma metáfora. Acreditamos que a própria linguagem por trás de XP pode ser usada na construção de uma metáfora que auxilia no aprendizado da metodologia, evoluindo junto com seu uso.

A **Propriedade Coletiva do Código** é um padrão simples de ensinar e adotar desde o início do aprendizado. Observamos um pouco de resistência proveniente dos **Velhos Dinossauros**, porém é importante que todos compreendam que quanto mais pessoas na equipe de desenvolvimento conhecerem todo o código, melhor. Nas experiências observadas equipes que tiveram que trocar desenvolvedores e não praticavam propriedade coletiva do código tiveram mais problemas. O anti-padrão **Não Sou Necessário Aqui** ocorre em contextos industriais e governamentais quando a prática de propriedade coletiva funciona tão bem que uma pessoa que se ausenta por questões pessoais ou de saúde sente-se desnecessária ao retornar e verificar que a equipe não sentiu sua falta [84]. Como solução recomendamos que o treinador converse com a pessoa e ajude-a a redescobrir seus objetivos pessoais dentro da organização.

As práticas de modelagem ágil são bons padrões para auxiliar no ensino de *design* incremental [5]. **Reunião de Modelagem** sugere que após o planejamento de um ciclo, desenvolvedores se encontrem e utilizem um **Quadro Branco** para pensar sobre a arquitetura do sistema, desenhando diagramas UML. Na Paggo após o planejamento de uma iteração no qual uma das histórias do cliente requeria mudanças arquiteturais no sistema, uma reunião de modelagem mostrou-se útil para que toda a equipe chegasse a um acordo sobre como evoluir a arquitetura. Tarefas específicas foram criadas para refatorar o sistema e implementar a nova arquitetura. No laboratório de XP equipes utilizam o quadro branco com frequência para desenhar diagramas de classe e seqüência antes de implementar funcionalidades. Na Cultura Digital os desenvolvedores mais experientes utilizaram algumas histórias de estudo para propor uma refatoração arquitetural do Estúdio Livre para o resto da equipe. Até mesmo pares podem se beneficiar de uma rápida reunião de modelagem antes de programar sua tarefa. O anti-padrão **Dono da Bola** explicita o problema que ocorre naturalmente quando somente o treinador, ou desenvolvedores mais experientes, usam a caneta para desenhar no quadro branco durante estas reuniões [84]. Como solução recomendamos que o treinador incentive os desenvolvedores menos experientes a anotar no quadro enquanto os mais experientes podem colaborar explicando conceitos e técnicas de UML.

O padrão **Refatoração** sugere que qualquer oportunidade de melhorar o código fonte de um

sistema seja explorada. Existem refatorações simples de ensinar, como as refatorações que envolvem renomear variáveis e métodos. Com o auxílio de ferramentas como o Eclipse, que automatizam os passos envolvidos em refatorações, alterando toda uma base de código, até refatorações mais complexas podem ser realizadas por desenvolvedores menos experientes. Porém observamos que a habilidade de reconhecer áreas no código onde é possível refatorar requer experiência. Sugerimos que a prática de refatorações simples seja adotada desde o início do desenvolvimento, refatorações mais complexas podem ser introduzidas ao longo do tempo.

Muitas vezes observamos áreas de uma aplicação que “cheiram mal” ou poderiam ser melhoradas, porém por alguma razão achamos melhor não refatorá-las imediatamente. O padrão **Limiar de Refatoração** sugere que anote-se em um radiador de informação dois limites para o número de refatorações pendentes (um que pode ser ultrapassado eventualmente e outro que se ultrapassado deve implicar na realização das refatorações pendentes) em um sistema de maneira com que a equipe sempre encontre-se confortável com a qualidade do código existente. O uso do Eclipse facilita a contabilização de refatorações pendentes, pode-se listar por exemplo todos os comentários que contém a *string* “TODO”.

O anti-padrão **O Chefe Refatorou o Nosso Código** se aplica em contextos governamentais e industrias quando existe um hierarquia entre os desenvolvedores [84]. Desenvolvedores menos experientes podem se sentir intimidados se os mais experientes refatorarem código criado por eles. Como solução recomendamos que se desenvolvedores mais experientes descobrirem oportunidades de refatorar código produzido por estagiários que esta oportunidade seja explorada com uma sessão de programação pareada onde aproveita-se para ensinar refatoração aos menos experientes.

Acreditamos que empresas e o governo devem investir em ao menos algumas pessoas bem qualificadas na equipe para garantir que a prática de *design* incremental e refatoração possam realmente ser aplicadas. Na academia é importante que os pré-requisitos da disciplina sejam claros, recomendamos que as práticas de modelagem ágil e refatoração sejam exploradas em disciplinas introdutórias (ie.: introdução a programação, introdução a programação orientada a objetos e tópicos de programação orientada a objetos) e que alunos com experiência no mercado participem das turmas. Em nossas observações a participação de pessoas com experiência mostrou-se essencial para que uma equipe aprenda ambas estas práticas.

#### 4.3.4 Padrões para aplicar no dia-a-dia

Algumas práticas de XP são propostas para acontecer diversas vezes ao dia, todos os dias. Vimos que **Integração Contínua** é importante para que uma versão funcional do sistema esteja sempre disponível. Acreditamos que esta prática é fácil de adotar no início do processo de aprendizagem. Ferramentas livres facilitam a vida de uma equipe de desenvolvimento providenciando repositórios de código fáceis de instalar e usar, além de automatizar o processo de verificar mudanças em um repositório ou integrar alterações feitas.

O padrão **Repositório Único** sugere que uma aplicação conte com somente um repositório, acreditamos que este padrão depende muito do contexto para ser aplicado ou não. Equipes que desenvolvem mais de um sistema podem ter mais de um repositório, além disso, muitas vezes um repositório com arcabouços comuns a mais de um sistema fazem sentido. Acreditamos que este padrão seja aplicado quando trabalhando com sistemas legados que estavam distribuídos em muitos repositórios. Quando sistemas legados estão sendo migrados vale a pena aplicar também o padrão **Implantação Incremental** que, como já vimos, sugere que ambos o novo sistema e o sistema legado fiquem disponíveis durante o período de migração, que deve acontecer em pequenos passos. Recomendamos o uso de ambos estes padrões quando lidando com sistemas legados para conseguir lidar com sua complexidade natural.

O processo de **Build Veloz** deve acontecer várias vezes ao dia. Vimos que o *script* de build é uma ferramenta essencial, porém não basta desenvolvê-lo no início do projeto. O investimento em aprimorar o *build* também deve ser contínuo. Tanto na Paggo quanto na Cultura Digital nossos *scripts* também eram responsáveis por instalar versões das aplicações em ambientes de teste e produção. Ferramentas livre como o CruiseControl podem auxiliar na tarefa de incrementar o build para que ele possa até baixar todo o código novo integrado em um dia, rodar todos os testes e mandar e-mails para os responsáveis com o resultado. Dominando estas tecnologias é até mesmo possível diminuir a frequência dos pequenos *releases* para que eles aconteçam diariamente, é o padrão **Implantação Diária**. Acreditamos que este padrão pode ser facilmente adotado se o sistema desenvolvido é um sistema para a web. Na Cultura Digital nosso sistema de *build* automaticamente publicava todas mudanças realizadas em um dia no ambiente de testes, que está disponível na web. Desta maneira usuários podem auxiliar no processo de testes, podendo realizar testes de aceitação manuais de uma funcionalidade no mesmo dia em que ela foi implementada. Talvez por contar com estes testes manuais a equipe tenha demorado tanto para realmente investir em testes automatizados.

Recomendamos que este tipo de práticas se complementem.

Vimos que **Programação Pareada** é um padrão amplamente reconhecido e validado. Um anti-padrão comum relativo a esta prática em contextos industriais e governamentais se chama **Prisão de Pareamento** e ocorre quando todo o tempo da equipe de desenvolvimento é reservado para programação pareada [84]. A solução é também separar algum tempo para as pessoas possam dedicar a tarefas pessoais. Na Paggo reservávamos 3 sessões de pareamento por dia, membros da equipe tinham uma hora de manhã e uma hora após o almoço para tratar de questões pessoais. Nestes contextos é também muito útil utilizar a programação pareada como ferramenta de seleção de novas pessoas na equipe. Na Paggo entrevistas são realizadas de forma que pretendentes a uma vaga participem de sessões de programação pareada com a equipe. Na academia recomendamos que o ensino desta técnica seja incluído em disciplinas introdutórias e incentivado através de trabalhos em pares. Outro anti-padrão relacionado a programação pareado se chama **Panelinha** e sugere que pares que nunca se revezam prejudicam o resto da equipe. Uma solução pode ser a adoção de uma “Matriz de Pareamento” como métrica para evidenciar este comportamento, além de incentivar o revezamento entre os pares.

O padrão **Desenvolvimento Dirigido por Testes** implica que testes sempre serão escritos para cobrir todo o código da aplicação. Na nossa experiência esta é uma das práticas mais difíceis de ensinar, tanto pela inexperiência de desenvolvedores quanto pelo fato de que defeitos são considerados normais em softwares. Infelizmente no contexto Brasileiro muitas vezes a qualidade não determina o sucesso. Muitas “gambiaras” permanecem como solução por preguiça e uma incrível capacidade dos brasileiros se adaptarem a dificuldades. Observamos um comportamento cético quanto aos testes na equipe da Cultura Digital, talvez por que defeitos sempre são evidenciados por testes manuais feitos pelos usuários dos sistemas. Recentemente a descoberta de um defeito de segurança grave fez com que alguns desenvolvedores começassem a compreender o valor de testes automatizados. Na Paggo nossos primeiros *releases* contavam com muitos defeitos. A prática de testes foi lentamente adotada. Observamos que ao longo do tempo, com mais experiência e mais testes desenvolvidos, os desenvolvedores começam a dar valor a esta prática. Acreditamos que a prática de testes é essencial. Recomendamos que o ensino de testes seja introduzido no início do curriculum acadêmico, inclusive podendo ser adotada para validação automática de exercícios programa.

Acreditamos que a dificuldade com os testes também é decorrência da falta de ferramentas que podem auxiliar o processo, até recentemente testes de aceitação de sistemas web eram muito complicados e trabalhosos de escrever. Atualmente, com ferramentas livres como o Selenium esperamos

### 4.3. APRENDIZADO ATRAVÉS DA REPETIÇÃO PRÁTICA - AMADURECIMENTO DA METODOLOGIA127

observar uma maior adoção de testes de aceitação automáticos.

Para evidenciar a necessidade de testes uma vez na Paggo observamos problemas com um dos *releases* e deixamos com que a equipe resolvesse entregá-lo ao cliente mesmo assim. Usamos a decepção do cliente ao ver o sistema com defeitos para reforçar a importância dos testes automatizados para a equipe.

O padrão **Análise de Causa Inicial** recomenda que após um defeito ser encontrado uma análise seja realizada para tentar determinar sua causa. Na maior parte dos casos a causa é falha humana. Este tipo de análise se mostra importante nas primeiras iterações de uma equipe aprendendo XP podendo explicitar deficiências na prática de testes, programação pareada, refatoração e propriedade coletiva do código.

O padrão **Código e Testes** sugere que estes sejam os únicos artefatos permanentes mantidos pela equipe. Para adotá-lo é importante que testes automatizados e *design* incremental estejam funcionando bem, recomendamos que este padrão só seja adotado após a equipe estar mais experiente e não ter medo de descartar outros artefatos gerados. Jogar fora os diagramas UML do *design* da arquitetura de um sistema apagando o quadro branco após uma reunião de modelagem pode incentivar a equipe a entendê-lo melhor e simplificar o código de maneira com que ele fique claro e comunique a intenção do *design*.

Um anti-padrão que pode emergir da prática diária de espaço aberto e informativo se chama **Espaço Aberto Demais** e foi observado em contextos industriais e governamentais. Uma equipe que comunica de forma visível seus progressos e seus defeitos pode ser considerada uma ameaça, ainda mais no contexto brasileiro governamental no qual, infelizmente, observamos o aparelhamento da máquina do estado e a incompetência de servidores indicados por razões políticas como norma. Ao tornar-se ágil e produtiva uma equipe atrai a inveja de outros setores ou departamentos de uma organização, críticas indevidas podem atrapalhar o fluxo de trabalho. Como solução a este anti-padrão recomendamos que o espaço seja aberto e comunicativo para dentro da equipe, mas conte com um certo isolamento do resto da organização. O padrão **Firewall** sugere que uma pessoa assuma o papel de intermediar a comunicação de uma equipe ágil com o resto da organização.

Finalmente, em contextos industriais quando observamos resistência contínua provendo de **Velhos Dinossauros** recomendamos o uso do padrão **Demita-o**. Durante o aprendizado na Paggo tornou-se claro que o desenvolvedor mais sênior não tinha interesse em aprender XP. Quando ele saiu da equipe observamos um avanço no aprendizado de todos. Desde então muitos funcionários que não se

adaptaram foram convidados a se retirar.

#### 4.4 Etapa final - A equipe desenvolve sua própria metodologia

Após usar a metodologia durante várias iterações a equipe deve ter adaptado as práticas de XP e até mesmo adotado novas práticas. É chegado o momento do meta-treinador se retirar e deixar a equipe guiar seu próprio processo de maneira independente. Existem alguns padrões que podem ajudar uma equipe nesta etapa final do seu aprendizado.

Um anti-padrão comum neste momento do aprendizado se chama **Complexo de Abandono** [56]. Este anti-padrão sugere solução para o problema comum de não alocar ninguém para o papel de treinador quando o meta-treinador deixa uma equipe. Quando isso acontece não recomendamos que alguém (possivelmente um desenvolvedor mais experiente) seja sobrecarregado com as tarefas do treinador. Até mesmo em contextos controlados como na academia observamos casos de alunos que atuavam como treinadores terem que desistir da disciplina. Também observamos que na maioria dos casos o desenvolvedor mais experiente não é a melhor escolha para preencher o papel de treinador. Como solução propomos que a equipe adote o padrão **Campeão da Corte** [75] elegendo uma ou mais pessoas (entre as que estão mais confortáveis com os valores, princípios e práticas de XP) para guiar a equipe no refinamento de sua metodologia de trabalho. Recomendamos ainda que mais de um campeão aceite essa responsabilidade e que o papel seja revezado entre membros da equipes como sugere o padrão **Treinador da Semana** [55].

Os padrões **Continuidade da Equipe** e **Redução da Equipe** sugerem que mantenha-se a equipe que está funcionando bem junta, porém se o projeto já amadureceu o suficiente a equipe pode ser reduzida sem perdas para a manutenção do sistema desenvolvido. Em uma organização grande sugerimos que pessoas que já passaram pelo processo de aprendizagem de XP em um projeto devem fazer parte de outras equipes que pretendem adotar a metodologia.

Se o projeto ainda precisa crescer o padrão **Em Fases** sugere que em um momento inicial desenvolvedores mais experientes sejam contratados e que em outros momentos desenvolvedores menos experientes sejam agregados ao time. Acreditamos que em qualquer caso onde novas pessoas precisam ser incorporadas a uma equipe o padrão **Time Auto-Selecionado** seja adotado. Este padrão sugere que a própria equipe participe do processo de seleção de novos membros (possivelmente através de sessões de programação pareada).

Acreditamos que o padrão **Retroperspectiva** é útil neste momento. Ele sugere que uma grande

retrospectiva seja realizada tentando observar todo um ano do processo, ao invés de sugerir ações imediatas que podem ser adotadas pela equipe a retrospectiva serve para que a equipe pense em perspectivas um pouco mais amplas e abstratas de como poderão evoluir no ano seguinte. Acreditamos que esta prática é valiosa para observar a quantidade de trabalho realizada em um grande período pela equipe, fazendo com que seja mais fácil para as pessoas abstraírem questões pessoais e sugerir melhorias levando em conta um contexto mais amplo.

#### 4.4.1 Observações Póstumas

Como parte do nosso trabalho também observamos como equipes se comportam depois de passar pela experiência de aprendizado de XP. No contexto acadêmico observamos que alunos não tem tempo de atingir a etapa de amadurecimento em uma única instância da disciplina. Por esta razão recomendamos que alunos interessados em se aprofundar em XP participem da disciplina por uma segunda vez, aceitando o papel de treinador em alguma das equipes. Observamos que alunos que se especializaram como treinadores obtêm mais experiência no uso da metodologia e estão aptos a guiar uma organização pela transição metodológica para XP. Na Paggo e na Cultura Digital realizamos visitas algum tempo após termos participado da experiência de ensino de XP para observar como as equipes estavam trabalhando.

O ambiente na Paggo demonstra como uma equipe pode construir um verdadeiro ambiente de aprendizado e melhoria contínua. Atualmente a empresa conta com 55 pessoas (incluindo um *call center*). Sendo que 8 pares trabalham na equipe de desenvolvimento base, 2 pares atuam na equipe de *web design* e 1 par atua como clientes *proxy* dentro da Oi, maior cliente da Paggo. Este par de clientes *proxy* é interessante pois cada um assume um papel, o **Tradutor** se responsabiliza por pegar requisitos gerados na Oi e traduzir para equipe de desenvolvimento dentro da Paggo e o **Cobrador** é responsável por garantir que pendências do lado do cliente serão resolvidas a tempo. O jogo do planejamento de cada grupo ocorre em uma única manhã, as iterações duram duas semanas e os desenvolvedores contam com 3 sessões de 2 horas diárias para programação pareada, além de uma hora reservada para ócio criativo. A cada iteração um dia é reservado para estudos e pesquisas, chama-se **Tech Day**. Estimativas são realizadas com precisão usando pontos relativos as sessões de programação pareada. Os desenvolvedores relatam que a adoção de software livre foi essencial para garantir ainda mais agilidade à equipe, por exemplo: eles utilizam a última versão disponível da base de códigos do MySQL (que não é considerada estável pela comunidade) sem medo e inclusive ajudaram a encontrar um defeito neste código. Vale comentar que a Paggo migrou de Oracle para

MySQL em apenas 3 semanas (isso adotando a prática de trabalho energizado). Outra observação que comprova a agilidade da equipe foi seu último contrato com a Oi, que teve o escopo reduzido em 20% e entrega antes da data prevista. Além disso, práticas ágeis são usadas em outros departamentos da empresa.

Já na Cultura Digital observamos que a equipe ainda está amadurecendo sua metodologia de trabalho. Notamos que esta experiência é peculiar por forçar a equipe a se re-estruturar ano após ano. Porém, neste último semestre, já sem apoio do treinador, a equipe adaptou sua metodologia de trabalho incluindo a maioria das práticas de XP e estendendo-as para funcionar em um ambiente de colaboração com atores distribuídos geograficamente. As iterações são semanais, contanto com um jogo do planejamento todas terças-feiras. A equipe estima suas tarefas em horas e não adota programação pareada sempre, porém com muita frequência e algumas vezes acontece até mesmo programação pareada remota. Para incluir desenvolvedores dispersos o jogo do planejamento conta com apoio de ferramentas como o Skype, streaming ao vivo de áudio de vídeo e uma sala de bate papo IRC. Além disso a equipe mantém um wiki onde todos aspectos relacionados as práticas metodológicas são armazenados, quando observam que emerge uma nova prática na equipe eles rapidamente atualizam o wiki para que todos desenvolvedores fiquem sabendo e também para que a comunidade de clientes saiba como estão trabalhando. A interação com a comunidade de usuários dos portais desenvolvidos é muito interessante, a prática de personas por exemplo evoluiu ao ponto de que a equipe utiliza personas reais baseadas nos usuários que são mais ativos na comunidade. Além disso a comunidade conta com o ambiente de desenvolvimento onde pode testar funcionalidades novas todos os dias (a atualização é automática), uma lista de discussão onde pode se relacionar com os desenvolvedores e ferramentas de listagem e acompanhamento de defeitos. Os desenvolvedores também escrevem relatórios mensais para explicar para a comunidade o que foi desenvolvido em um *release* e mantêm um *blog* para relatar suas atividades e descobertas diárias. Entre as duas novas práticas que mais nos interessou estão a **Imersão eXtrema** e o **Tracker Bot**. Na **Imersão eXtrema** toda equipe se encontra em um ambiente externo no qual, durante uma semana, trabalham e convivem juntos, todos os dias, o dia todo. Este padrão se mostrou muito efetivo para integrar novos membros na equipe. Acreditamos que ele aumenta consideravelmente a capilaridade dos fluxos de comunicação de uma equipe, além de propiciar um ambiente de alta criatividade e produção, aumentando as possibilidades de encontrar-se defeitos e oportunidades para refatoração e simplificação do design de uma aplicação. O padrão **Tracker Bot** sugere que quando a equipe domina o papel de acompanhador que suas funções sejam automatizadas. Na Cultura Digital um robô foi programado para estar sempre

*on-ine* no canal IRC, recebendo informações do repositório de código fonte e coletando estatísticas sobre o processo de desenvolvimento, é esse robô que gera as métricas usadas pela equipe para avaliar o andamento do seu processo. Na Cultura Digital algumas práticas ágeis também são usadas por equipes não ligadas ao desenvolvimento de software.

#### 4.5 Práticas Ágeis em contextos não ligados ao desenvolvimento de software

Observamos que algumas práticas ágeis podem trazer benefícios até mesmo para grupos que não estão programando. Acreditamos que os valores e princípios da Programação eXtrema podem ser aplicados em outros contextos de uma organização e observamos duas experiências práticas do gênero. Tanto na Paggo quanto na Cultura Digital outras equipes adotaram alguns dos padrões das metodologias ágeis para melhorar a maneira como trabalhavam.

Os valores e princípios de XP são genéricos o suficiente para serem aplicados em contextos além da programação de software. Acreditamos que comunicação, *feedback*, coragem e respeito são valores que podem melhorar qualquer tipo de trabalho. Os princípios que são a base das práticas de XP também podem ser usados para criar práticas de trabalho em outras áreas.

Em particular observamos o valor de documentar requisitos como histórias. No primeiro ano do projeto da Cultura Digital utilizamos histórias para explicitar requisitos das diferentes equipes, a cada quinze dias representantes destas equipes se reuniam e apresentavam aos outros as histórias que tinham realizado e o *backlog* de pendências, durante esta reunião equipes poderiam sugerir histórias as outras. Acreditamos que esta experiência foi válida para aumentar a comunicação em um grande projeto com uma equipe de 80 pessoas distribuídas pelo país. Nos quase dois meses nos quais praticamos histórias nas diferentes áreas do projeto acreditamos que elas foram valiosas para que todas as equipes tivessem uma concepção mínima do trabalho sendo realizado pelo resto da organização, a prática melhorou a comunicação e a contabilidade entre as diferentes equipes. Infelizmente ao final do ano o projeto passou por uma reformulação que criou o cargo de “coordenador” nas diferentes áreas o que criou um gargalo na comunicação e acabou centralizando o processo de comunicação do grupo, tentamos manter a prática de compartilhar histórias e fornecer *feedback* sobre o progresso nas diferentes áreas em uma lista de e-mails para a coordenação, porém esta tentativa não foi bem sucedida. Hoje em dia as diferentes equipes não tem uma percepção clara do trabalho realizados pelos outros, por mais que algumas equipes se esforcem para relatar no que trabalhou usando relatórios mensais e blogs. Na Paggo atualmente todas as áreas utilizam histórias para especificar seus projetos, em áreas como marketing ou relações com o cliente as histórias contam com estimativas abstratas,

podendo ser relativas a uma tarefa de cobrança de um colaborador após um tempo ou até mesmo marcando a data de quando alguém de fora da organização prometeu dar o retorno necessário para o andamento de um projeto. Todas as áreas contam com um quadro de histórias e cada história tem dois responsáveis. Em um jogo do planejamento que ocorre a cada duas semanas as diferentes áreas podem verificar quais tarefas foram realizadas e quais ainda estão pendentes. Se um desenvolvedor precisa de algo relacionado a um projeto da equipe de marketing ele pode facilmente olhar o quadro de histórias desta equipe para saber do andamento do projeto.

O uso da prática de histórias também implica no uso de versões modificadas das práticas de ciclos e jogos do planejamento. Além disso, o papel do Cliente mostrou-se adaptável a outros contextos e as histórias são escritas por pessoas que assumem este papel nas diferentes áreas das organizações.

A prática de planejamento ágil mostrou-se uma excelente aliada para organização de eventos onde a equipe se encontrava para trabalhar em algum produto específico. Além disso, na organização de oficinas para os pontos de cultura o planejamento ágil foi muito utilizado para que pudéssemos lidar com as eventuais mudanças de horários ou até mesmo propostas de novas atividades feitas por pessoas dos pontos. A Figura 4.3 mostra um mural com o resultado do planejamento ágil de uma oficina da Cultura Digital.

Programação pareada também é uma prática que pode ser adaptada para outros conceitos. Na Cultura Digital por exemplo a equipe de produção de vídeos experimentou fazê-lo de modo com que duas pessoas trabalhassem juntas. Na Paggo todas tarefas de qualquer projeto sempre contam com dois responsáveis e são executadas, na sua grande maioria, em pares de modo que se alguém não estiver presente na empresa existe sempre uma pessoa que está acompanhando o trabalho sendo realizado.

A prática de papo-em-pé também se mostrou facilmente adaptável à outros contextos. Durante as “Oficinas de Conhecimentos Livres” da Cultura Digital muitas vezes a equipe se encontrava de manhã e realizava um papo-em-pé para saber melhor como estava a oficina. Na Paggo todas as áreas se encontram na frente do quadro de histórias para discutir as tarefas que devem realizar um dia. Observamos que esta prática é muito efetiva para melhorar a comunicação interna em equipes que colaboram no seu trabalho.

Finalmente a prática de retrospectivas e retoperspectivas mostra-se valiosa independente da área na qual é aplicada. Tanto na Paggo quanto na Cultura Digital retrospectivas são realizadas com frequência para que equipes possam descobrir falhas e propor melhorias ao seu processo de trabalho.

#### 4.5. PRÁTICAS ÁGEIS EM CONTEXTOS NÃO LIGADOS AO DESENVOLVIMENTO DE SOFTWARE<sup>133</sup>



Figura 4.3: Um mural mostra o planejamento ágil de uma oficina de conhecimentos livres. Participantes podiam propor mudanças no cronograma e até mesmo novas atividades durante a oficina.

A Figura 4.4 mostra um cartaz de uma retrospectiva realizada por todas as áreas da Cultura Digital em conjunto.



Figura 4.4: Retrospectivas são úteis para qualquer equipe que trabalha de maneira colaborativa.

## Capítulo 5

### Conclusões

Este trabalho estudou o ensino de XP como metodologia que pode ser aplicada em contextos acadêmicos, industriais e governamentais. Além da discussão sobre os conceitos e teorias relacionados ao ensino e aplicação de XP, três estudos de caso foram conduzidos. Analisamos estes casos e também evidências presentes na literatura para validar alguns padrões que podem ser usados para facilitar o ensino da metodologia nestes diferentes contextos.

Apresentamos um pouco da história e motivação por trás do surgimento das metodologias ágeis e refletimos sobre os valores, princípios e práticas propostos pela Programação eXtrema, observando a evolução pela qual passou a metodologia ao longo dos últimos anos.

Mostramos que o ensino de XP não é uma tarefa complexa, porém também não é trivial. Descrevemos diversas experiências de ensino e adoção da metodologia relatadas pela comunidade ágil e entramos em detalhes sobre nossas experiências no Laboratório de Programação eXtrema, na Paggo e na Cultura Digital.

Analisamos as experiências relatadas e refletimos sobre elas, sugerindo padrões que podem facilitar o ensino da metodologia nos diferentes contextos. Acreditamos que os padrões apresentados podem facilitar o ensino de XP, sendo que alguns são específicos para serem aplicados na academia, na indústria ou no governo.

Acreditamos que ensinar XP é de extrema importância para melhorar a qualidade do software que está sendo produzido no Brasil. A tarefa é simples e rápida se comparada ao ensino de metodologias tradicionais e os padrões apresentados neste trabalho podem ser utilizados como ferramentas para auxiliar o educador que embarca nesta empreitada.

## 5.1 Principais contribuições

Dentre as principais contribuições deste trabalho, destacam-se:

- O estudo detalhado de experiências de ensino na academia, indústria e governo presentes na literatura.
- O estudo de três experiências práticas de ensino da metodologia.
- A sugestão de padrões que podem melhorar a experiência de ensino em diversos contextos, separados entre aqueles que podem ser aplicados antes de começar uma experiência, durante o ensino e após uma equipe amadurecer sua prática metodológica.
- Uma primeira experiência de uso de práticas ágeis em outras áreas que não a programação de software.

## 5.2 Trabalhos futuros

Acreditamos que o presente estudo pode ser melhorado, sugerimos os seguintes tópicos como questões que podem ser aprimoradas no futuro:

- Atualizar a pesquisa da literatura acadêmica, levando em conta artigos mais atuais que não tivemos tempo hábil de pesquisar, escritos a partir de 2006.
- Sugerir mais padrões e tecnologias pra melhorar o aprendizado da prática de testes automatizados, pois observamos que esta prática é a que mais apresenta problemas.
- Realizar mais experiências em contextos governamentais para explicitar melhor as diferenças entre este contexto e a industria.
- Fazer um trabalho detalhado sobre o uso de práticas ágeis em grupos que não tem como objetivo final programar software.

## Referências Bibliográficas

- [1] Joichi Abe, Ken Sakamura, and Hideo Aiso, *An analysis of software project failure*, ICSE '79: Proceedings of the 4th international conference on Software engineering (Piscataway, NJ, USA), IEEE Press, 1979, pp. 378–385. 2
- [2] C. Alexander, S. Ishikawa, M. Silverstein, et al., *A pattern language: towns, buildings, construction*, Oxford University Press, 1977. 12, 110
- [3] GB Alleman, M. Henderson, and R. Seggelke, *Making agile development work in a government contracting environment-measuring velocity with earned value*, Agile Development Conference, 2003. ADC 2003. Proceedings of the (2003), 114–119. 85
- [4] Agile Alliance, *Manifesto for Agile Software Development*, Home page: <http://agilemanifesto.org>, 2001. 5
- [5] S. W. Ambler, *Agile modeling*, John Wiley & Sons, 2002. 123
- [6] Scott W. Ambler and Pramod J. Sadalage, *Refactoring databases: Evolutionary database design*, Addison-Wesley, 2006. 9, 30
- [7] S.W. Ambler, *Process Patterns-Buiding Large-Scale Systems Using OO Technology*, 1998. 12, 100
- [8] ———, *More process patterns*, SIGS Books New York, 1999. 12, 100
- [9] Johan Andersson, Geoff Bache, and Peter Sutton, *XP with acceptance-test driven development: A rewrite project for a resource optimization system*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Notes on Computer Science, vol. 2675, 2003, pp. 180–188. 9
- [10] J. Andrea, *Managing the Bootstrap Story in an XP Project*, Proceedings of XP 2001 (North Carolina,), 2001. 9, 115
- [11] Gary H. Anthes, *Sabre takes extreme measures*, Computer World, see [www.computerworld.com/softwaretopics/software/story/0,10801,91646,00.html](http://www.computerworld.com/softwaretopics/software/story/0,10801,91646,00.html), March 2004. 9, 53

- [12] G. Asproni, *Motivation, teamwork, and agile development*, Agile Times **IV** (2004), no. 1, 8–15. [8](#)
- [13] O. Astrachan, R. Duvall, and E. Wallingford, *Bringing extreme programming to the classroom*, Proceedings of XP Universe 2001 (Raleigh, NC, USA), 2001. [9](#), [55](#)
- [14] B. Aveling, *XP Lite Considered Harmful?*, Proceedings of the 5th international conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, Springer, 2004, pp. 94–103. [9](#), [53](#)
- [15] K. Beck and E. Gamma, *Test infected: Programmers love writing tests*, Java Report **3** (1998), no. 7, 37–50. [9](#)
- [16] Kent Beck, *Extreme programming explained: Embrace change*, 1 ed., Addison-Wesley, 1999. [8](#), [11](#), [24](#), [36](#), [62](#), [100](#), [116](#)
- [17] ———, *The metaphor metaphor*, Keynote speech - ACM OOPSLA'02, November 2002. [20](#)
- [18] ———, *Test-driven development: By example*, Addison-Wesley, 2002. [9](#), [29](#)
- [19] Kent Beck and Cynthia Andres, *Extreme programming explained: Embrace change, 2nd edition*, 2 ed., Addison-Wesley, 2004. [4](#), [8](#), [11](#), [20](#), [41](#), [99](#), [100](#), [108](#), [109](#)
- [20] P. Becker-Pechau, H. Breitling, M. Lippert, and A. Schmolitzky, *Teaching Team Work: An Extreme Week for First-Year Programmers*, Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Notes on Computer Science, vol. 2675, Springer, 2003, pp. 386–393. [9](#), [56](#)
- [21] Yochai Benkler, *Coase's penguin, or, linux and the nature of the firm*, Yale Law Journal **1** (2002), no. 112, 369–446. [5](#)
- [22] L. Bossavit, *The Unbearable Lightness of Programming: A Tale of Two Cultures*, CUTTER IT JOURNAL **15** (2002), no. 9, 5–11. [9](#)
- [23] P. Bossi, *Using Actual Time: Learning How to Estimate*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Notes on Computer Science, vol. 2675, 2003, pp. 244–253. [9](#)
- [24] Piergiuliano Bossi, *eXtreme Programming applied: a case in the private banking domain*, Proceedings of OOP (Munich), 2003, Available at [www.quinary.com/pagine/downloads/files/Resources/OOP2003Paper.pdf](http://www.quinary.com/pagine/downloads/files/Resources/OOP2003Paper.pdf). [9](#), [53](#), [78](#)
- [25] H.W. McCormick Brown, W.J. and S.W. Thomas, *AntiPatterns in Project Management*, John Wiley & Sons, 2000. [12](#), [100](#)

- [26] B.G. Cain, J.O. Coplien, and N.B. Harrison, *Social patterns in productive software development organizations*, Annals of Software Engineering **2** (1996), no. 1, 259–286. [12](#), [100](#)
- [27] Jan Chong, *Social behaviors on xp and non-xp teams: A comparative study*, ADC '05: Proceedings of the Agile Development Conference (Washington, DC, USA), IEEE Computer Society, 2005, pp. 39–48. [8](#)
- [28] A. Cockburn and L. Williams, *The costs and benefits of pair programming*, Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000) (Cagliari, Sardinia, Italy), June 2000. [9](#), [30](#)
- [29] Alistair Cockburn, *Agile software development: The cooperative game*, Addison-Wesley Longman, 2002. [8](#), [12](#), [20](#), [40](#), [60](#), [117](#)
- [30] ———, *Crystal clear: A human-powered methodology for small teams*, Addison-Wesley Professional, 2004. [8](#)
- [31] J.O. Coplien and N.B. Harrison, *Organizational Patterns of Agile Software Development*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2004. [12](#), [100](#), [109](#), [110](#), [116](#)
- [32] Robert Cowham and Matt Stephens, *To xp or not to xp?*, The Computer Bulletin **47** (2005), no. 2, 16–16. [9](#)
- [33] L. Crispin, T. House, and C. Wade, *The Need for Speed: Automating Acceptance Testing in an Extreme Programming Environment*, Second International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP 2001 (2001), 96–104. [9](#)
- [34] Ron Crocker, *Large-scale agile software development*, Addison-Wesley, 2004. [50](#)
- [35] Mark Crowne, *Why software product startups fail and what to do about it*, Proceedings of 2002 IEEE International Engineering Management Conference (IEMC 2002), 2002, pp. 338–343. [2](#)
- [36] E.G. da Costa Filho, R. Penteado, J.C.A. Silva, and R.T.V. Braga, *Padrões e Métodos Ágeis: agilidade no processo de desenvolvimento de software*, Proceedings of the 5th Latin American Conference on pattern Languages of Programming, 2005, pp. 156–169. [13](#), [100](#)
- [37] Dicionário Houaiss da Língua Portuguesa, *Metodologia*, verbete, p. 1911. [12](#)
- [38] Novo Dicionário Básico da Língua Portuguesa Aurélio, *Metodologia*, verbete, p. 431. [12](#)
- [39] Alfredo Goldman Danilo Sato, Dairton Bassi, *Extending extreme programming with practices from other agile methodologies*, Workshop de Desenvolvimento Rápido de Aplicações (WDRA 2007), 2007. [39](#), [41](#), [121](#)
- [40] Sérgio Buarque de Holanda, *Raízes do brasil*, Companhia das Letras, 1995. [100](#)

- [41] Domenico de Masi, *O Ócio Criativo*, Sextante, Rio de Janeiro, 2000. 3
- [42] Roberto Deias, Giampiero Mugheddu, and Orlando Murru, *Introducing xp in a start-up*, Proceedings of the 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2002), 2002. 9, 53, 75
- [43] Joel A. Oroszi, Doug Wallace, Isobel Raggett, *Extreme Programming for Web Projects*, Addison-Wesley, 2003. 8
- [44] Y. Dubinsky, *Teaching eXtreme Programming in a Project-Based Capstone Course*, Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), vol. 2675, Springer, 2003, pp. 435–436. 9, 57
- [45] Yael Dubinsky and Orit Hazzan, *Roles in agile software development teams*, Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, 2004, pp. 157–165. 9, 57, 110
- [46] Nicholas Epley and Justin Kruger, *Egocentrism over e-mail: Can we communicate as well as we think?*, *Journal of Personality and Social Psychology* **89** (2006), no. 5, 925–936. 20
- [47] C. Farrell, R. Narang, S. Kapitan, and H. Webber, *Towards an effective onsite customer practice*, Proceedings of the 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2002), 2002, pp. 52–55. 9
- [48] Ernst Fehr and Simon Gächter, *Do Incentive Contracts Crowd Out Voluntary Cooperation?*, Institute for Empirical Research in Economics, University of Zurich, Working Paper No. 34 <http://www.iew.unizh.ch/wp/iewwp034.pdf>, 2002. 5
- [49] Ernst Fehr and John A. List, *The hidden costs and returns of incentives - trust and trustworthiness among ceos.*, *Journal of the European Economic Association* **2** (2004), no. 5, 743–771. 5
- [50] Martin Fowler, *Refactoring: Improving the design of existing code*, Addison-Wesley, 1999. 9, 29
- [51] ———, *Is design dead?*, Proceedings of the Extreme Programming Conference 2000 Calgary, Italy (XP2000), 2000, Revised 2004 version available at <http://www.martinfowler.com/articles/designDead.html>. 4
- [52] ———, *Yesterday's weather*, available at: <http://www.martinfowler.com/bliki/YesterdaysWeather.html>, 2001. 24
- [53] Alexandre Freire, Francisco Gatto, and Fábio Kon, *Cigarra-A Peer-to-Peer Cultural Grid*, Anais do 6o Workshop sobre Software Livre (WSL 2005), 2005, pp. 177–183. 8, 58, 59, 72

- [54] Alexandre Freire, Alfredo Goldman, Carlos Eduardo Ferreira, Christian Asmussen, and Fábio Kon, *Mico - university schedule planner*, Anais do 5o Workshop sobre Software Livre (WSL 2004) (Porto Alegre), Junho 2004, pp. 147–150. [8](#), [58](#), [69](#)
- [55] Alexandre Freire, Fábio Kon, and Cícero Torteli, *Xp south of the equator: An experience implementing xp in brazil*, Proceedings of the XP 2005 Conference, Lecture Notes on Computer Science, vol. 3556, Springer, 2005, pp. 10–18. [9](#), [79](#), [121](#), [122](#), [128](#)
- [56] Alexandre Freire, Fábio Kon, and Alfredo Goldman, *The “bootstrap”, “split personality” and “abandon complex” antipractices - experiences teaching xp in academic and business environments*, Workshop de Desenvolvimento Rápido de Aplicações (WDRA 2007), 2007. [12](#), [100](#), [108](#), [115](#), [128](#)
- [57] Alexandre Freire and Paulo Silveira, *Vraptor - simple and quick web framework*, Anais do 5o Workshop sobre Software Livre (WSL 2004), 2004, pp. 39–42. [8](#)
- [58] Bruno S. Frey and Reto Jegen, *Motivation crowding theory: A survey of empirical evidence.*, Journal of Economic Surveys **15** (2001), 589–611. [5](#)
- [59] Bruno S. Frey and Felix Oberholzer-Gee, *The cost of price incentives: An empirical analysis of motivation crowding out*, American Economic Review **87** (1997), 746–755. [5](#)
- [60] Andrew M. Fuqua and John M. Hammer, *Embracing change: An XP experience report*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Notes on Computer Science, vol. 2675, 2003, pp. 298–306. [9](#), [53](#), [77](#)
- [61] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns*, Addison-Wesley, Reading, MA, 1995. [12](#), [30](#)
- [62] B. George and L. Williams, *A Structured Experiment of Test-Driven Development*, Information and Software Technology **46** (2004), no. 5, 337–342. [9](#), [29](#)
- [63] Alfredo Goldman, Fábio Kon, Paulo J. S. Silva, and Joseph W. Yoder, *Being Extreme in the Classroom: Experiences Teaching XP*, Journal of the Brazilian Computer Society **10** (2004), no. 2, 1–17. [7](#), [9](#), [57](#), [58](#), [86](#)
- [64] L. Griffin, *A customer experience: Implementing xp*, Proceedings of the XP Agile Universe Conference, 2001. [9](#)
- [65] O. Hazzan and Y. Dubinsky, *Teaching a software development methodology: the case of extreme programming*, Proceedings of the 16th Conference on Software Engineering Education and Training (CSEE&T 2003), 2003, pp. 176–184. [9](#), [57](#)

- [66] Orit Hazzan, Joe Bergin, James Caristi, Yael Dubinsky, and Laurie Williams, *Teaching software development methods: The case of Extreme Programming*, Panel at the 35th Technical Symposium on Computer Science Education (SIGCSE'2004) (Norfolk, Virginia, USA), ACM, 2004, <http://db.grinnell.edu/sigcse/sigcse2004/viewAcceptedSession.asp?sessionType=SpecialSession&sessionNumber=28>. 9, 54
- [67] G. Hedin, L. Bendix, and B. Magnusson, *Coaching Coaches*, Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), vol. 2675, Springer, 2003, pp. 154–160. 9, 57
- [68] ———, *Introducing software engineering by means of extreme programming*, Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 586–593. 9, 57
- [69] ———, *Teaching extreme programming to large groups of students*, The Journal of Systems & Software 74 (2005), no. 2, 133–146. 9, 57
- [70] M. Holcombe, M. Gheorghe, and F. Macias, *Teaching xp for real: Some initial observations and plans*, Proceedings XP 2001, 2001, pp. 14–17. 9, 56
- [71] Donald Howard, *Swimming around the waterfall: Introducing and using agile development in a data centric, traditional software engineering company*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Note in Computer Science, vol. 2675, 2003, pp. 138–145. 9, 53, 77
- [72] Hanna Hulkko and Pekka Abrahamsson, *A multiple case study on the impact of pair programming on product quality*, ICSE '05: Proceedings of the 27th international conference on Software engineering, ACM Press, 2005, pp. 495–504. 9, 22
- [73] Andrew Hunt and David Thomas, *The pragmatic programmer*, Addison-Wesley, 2000. 8
- [74] J.A. Highsmith III, *Adaptive software development: A collaborative approach to managing complex systems*, Dorset House Publishing, New York, 1999. 8
- [75] et al. Jackson, A., *Behind the Rules: XP Experiences*, Proceedings of the 2004 Agile Development Conference (Salt Lake City), 2004. 9, 53, 128
- [76] Ron Jeffries, *Big visible charts*, available at <http://www.xprogramming.com/xpmag/BigVisibleCharts.htm>, 2004. 38
- [77] K. Johansen, R. Stauffer, and D. Turner, *Learning by Doing: Why XP Doesn't Sell*, Proceedings of the XP Agile Universe 2001 Conference, Addison Wesley Longman, 2001. 9, 53
- [78] G. Keefer, *Extreme Programming Considered Harmful for Reliable Software Development*, AVOCA [Advanced Visioning of Components and Architectures] Technical Report (2002). 9

- [79] Roy Miller Ken Auer, *Extreme Programming Applied: Playing to Win*, Addison-Wesley, 2002. 8
- [80] Martin Fowler Kent Beck, *Planning Extreme Programming*, Addison-Wesley, 2001. 8, 24
- [81] Joshua Kerievsky, *Refactoring to patterns*, Addison-Wesley, 2005. 29
- [82] J. Koskela and P. Abrahamsson, *On-Site Customer in an XP Project: Empirical Results from a Case Study*, at European Software Process Improvement Conference (EuroSPI 2004), Trondheim, Norway, Lecture Note in Computer Science, vol. 3281, Springer, 2004, pp. 1–11. 9
- [83] S. Kuppuswami, K. Vivekanandan, Prakash Ramaswamy, and Paul Rodrigues, *The effects of individual xp practices on software development effort*, SIGSOFT Softw. Eng. Notes **28** (2003), no. 6, 6–6. 8
- [84] Y. Kuranuki and K. Hiranabe, *AntiPractices: AntiPatterns for XP Practices*, Proceedings of the 2004 Agile Development Conference (Salt Lake City), 2004. 12, 100, 117, 123, 124, 126
- [85] P. Lappo, *No pain, no XP: observations on teaching and mentoring extreme programming to university students*, Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002) (Cagliari, Italy), 2002. 9, 55
- [86] L. Layman, L. Williams, and L. Cunningham, *Exploring extreme programming in context: an industrial case study*, Agile Development Conference, 2004 (2004), 32–41. 9, 53, 77
- [87] Lucas Layman, *Empirical investigation of the impact of extreme programming practices on software projects*, OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (New York, NY, USA), ACM Press, 2004, pp. 328–329. 8
- [88] Lucas Layman, Laurie Williams, and Lynn Cunningham, *Motivations and measurements in an agile case study*, QUTE-SWAP '04: Proceedings of the 2004 workshop on Quantitative techniques for software agile process (New York, NY, USA), ACM Press, 2004, pp. 14–24. 8
- [89] Tip House Lisa Crispin, *Testing Extreme Programming*, Addison-Wesley, 2003. 8
- [90] Jim Little, *Ats diary*, C2 wiki site, 2000, see <http://c2.com/cgi/wiki?AtsDiary>. 9, 53
- [91] Kim M. Lui and Keith C. C. Chan, *Pair programming productivity: Novice-novice vs. expert-expert*, Int. J. Hum.-Comput. Stud. **64** (2006), no. 9, 915–925. 9
- [92] Kim Man Lui and Keith C.C. Chan, *Test driven development and software process improvement in china*, Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, 2004, pp. 219–222. 9, 29

- [93] K.M. Lui and K.C.C. Chan, *When Does a Pair Outperform Two Individuals?*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Note in Computer Science, vol. 2675, Springer, 2003. 9, 22, 30
- [94] K. Mannaro, M. Melis, and M. Marchesi, *Empirical Analysis on the Satisfaction of IT Employees Comparing XP Practices with Other Software Development Methodologies*, Proceedings of the 5th international conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, Springer, 2004, pp. 166–174. 8
- [95] A. Martin, J. Noble, and R. Biddle, *Being Jane Malkovich: A Look Into the World of an XP Customer*, Proceedings of the 4 thInternational Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2003), Lecture Note in Computer Science, vol. 2675, Springer, 2003, pp. 234–243. 9
- [96] Angela Martin, *Exploring the XP Customer Role ? Part II*, Proceedings of the 5th international conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, Springer, 2004, p. 318. 9
- [97] Angela Michelle Martin, *The role of customers on extreme programming projects*, Ph.D. thesis, Victoria University of Wellington, Department of Philosophy, 2007. 9
- [98] R. Biddle Martin, A. and J. Noble, *The XP Customer Role in Practice: Three Studies*, Proceedings of the 2004 Agile Development Conference (Salt Lake City), 2004. 9
- [99] K. Maruyama and K. Shima, *Automatic method refactoring using weighted dependence graphs*, Proceedings of the 21st international conference on Software engineering, IEEE Computer Society Press, 1999, pp. 236–245. 9
- [100] P. McBreen, *Questioning extreme programming*, Addison Wesley, 2003. 9
- [101] D. McKinney, J. Froeseth, J. Robertson, L.F. Denton, and D. Ensminger, *Agile CS1 Labs: eXtreme Programming Practices in an Introductory Programming Course*, Proceedings of XP/Agile Universe 2004, Calgary, Canada, August 15-18, 2004. Lecture Notes in Computer Science (2004). 9, 57
- [102] Grigori Melnik and Frank Maurer, *Comparative analysis of job satisfaction in agile and non-agile software development teams.*, Proceedings of Extreme Programming and Agile Processes in Software Engineering, 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Lecture Notes in Computer Science, vol. 4044, Springer, 2006, pp. 32–42. 8
- [103] Atif M. Memon, *A Comprehensive Framework for Testing Graphical User Interfaces*, Ph.D. thesis, University of Pittsburgh, July 2001. 9

- [104] ———, *GUI testing: Pitfalls and process*, IEEE Computer **35** (2002), no. 8, 90–91. [9](#)
- [105] J. Meszaros, *Using Storyotypes to Split Bloated XP Stories*, Proceedings of the XP/Agile Universe 2004 (North Carolina,), vol. 3134, Springer, 2004, pp. 73–80. [9](#), [115](#)
- [106] Vojislav B. Misić, *Perceptions of extreme programming: an exploratory study*, SIGSOFT Softw. Eng. Notes **31** (2006), no. 2, 1–8. [8](#)
- [107] Yasuhiro Monden, *Toyota production system: An integrated approach to just-in-time*, 3 ed., Engineering & Management Press, Norcross, GA, 1998. [5](#)
- [108] R. Mugridge, *Challenges in Teaching Test Driven Development*, Proceedings of the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2003), Lecture Notes on Computer Science, vol. 2675, 2003, pp. 410–413. [9](#)
- [109] R. Mugridge, B. MacDonald, P. Roop, and E. Tempero, *Five challenges in teaching xp*, Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), vol. 2675, Springer, 2003, pp. 406–409. [9](#), [57](#)
- [110] M. Müller, J. Link, R. Sand, and G. Malpohl, *Extreme programming in curriculum: Experiences from academia and industry*, Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), LNCS, vol. 3092, Springer, June 2004, pp. 294–302. [9](#), [57](#)
- [111] M. Müller and W. Tichy, *Case study: extreme programming in a university environment*, Proceedings of the 23rd International Conference on Software Engineering, 2001. ICSE 2001, 2001, pp. 537–544. [9](#), [57](#)
- [112] M.M. Muller and F. Padberg, *About the return on investment of test-driven development*, International Workshop on Economics-Driven Software Engineering Research EDSER **5** (2003). [9](#)
- [113] Roger A. Müller, *Extreme Programming in a university project*, Proceedings of the 5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, 2004, pp. 312–315. [9](#), [57](#)
- [114] N. Nagappan, L. Williams, E. Wiebe, C. Miller, S. Balik, M. Ferzli, and M. Petlick, *Pair learning: With an eye toward future success*, Extreme Programming and Agile Methods - XP/Agile Universe 2003, Lecture Notes in Computer Science, vol. 2753 / 2003, Springer-Verlag Heidelberg, September 2003, pp. 185 – 198. [9](#)
- [115] M.F. Nisar and T. Hameed, *Agile methods handling offshore software development issues*, Proceedings of the 8th International Multitopic Conference (INMIC 2004), 2004, pp. 417– 422. [40](#)

- [116] J. Noble, S. Marshall, S. Marshall, and R. Biddle, *Less Extreme Programming*, Proceedings of the sixth conference on Australian computing education **30** (2004), 217–226. 9, 56
- [117] John Noll and Darren C Atkinson, *Comparing extreme programming to traditional development for student projects: A case study*, Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003, Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 372–374. 9, 55
- [118] Steven R. Palmer and John M. Felsing, *A practical guide to feature-driven development*, Prentice Hall, 2002. 8
- [119] Joseph Pelrine, *Modelling infection scenarios - a fixed-price extreme programming success story*, ACM OOPSLA Companion Proceedings, ACM Press, 2000, pp. 23–24. 9, 53, 75
- [120] D. E. Perry and W. M. Evangelist, *An empirical study of software interface faults*, Proceedings of the International Symposium on New Directions in Computing, August 1985, pp. 32–38. 2
- [121] M. Poppendieck and T. Poppendieck, *Lean software development: An agile toolkit for software development managers*, Addison-Wesley, 2003. 8
- [122] D. J. Reifer, *How to get the most out of extreme programming/agile methods*, Proceedings of the 2nd XP and 1st Agile Universe Conference (Chicago, IL), Lecture Notes on Computer Science, vol. 2418, Springer, August 2002, pp. 185–196. 8
- [123] L. Rising and E. Derby, *Singing the Songs of Project Experience: Patterns and Retrospectives*, Journal of Information Technology **16** (2003), no. 9. 40
- [124] Linda Rising and Mary Lynn Manns, *Fearless change. patterns for introducing new ideas*, Addison-Wesley, Boston, 2005. 12, 99, 100
- [125] R.O. Rogers, *Scaling Continuous Integration*, Proceedings of the 5th international conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2004), Lecture Notes on Computer Science, vol. 3092, Springer, 2004, pp. 68–76. 9
- [126] Chet Hendrickson Ron Jeffries, Ann Anderson, *Extreme Programming Installed*, Addison-Wesley, 2001. 8
- [127] B. Rumpe and A. Schröder, *Quantitative survey on extreme programming projects*, Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002) (Alghero, Italy), 2002, pp. 95–100. 8
- [128] O. Salo, and P. Kyllönem K. Kolehmainen, J. Löthman, S. Salmijärvi, and P. Abrahamsson, *Self-adaptability of agile software processes: A case on post-iteration workshops*, Proceedings of the 5th international conference on eXtreme Programming and Agile Processes in Software

- Engineering (XP 2004), Lecture Notes in Computer Science, vol. 3092, Springer, 2004, pp. 184–193. 40
- [129] Outi Salo and Pekka Abrahamsson, *Empirical evaluation of agile software development: The controlled case study approach*, Proceedings of the 5th International Product Focused Software Process Improvement Conference (PROFES 2004), Lecture Notes in Computer Science, vol. 3009, Springer Berlin / Heidelberg, 2004, pp. 408–423. 8
- [130] Danilo Sato, Dairton Bassi, Mariana Bravo, Alfredo Goldman, and Fabio Kon, *Experiences tracking agile projects: an empirical study*, Journal of the Brazilian Computer Society (2007), <http://www.dtsato.com/resources/default/jbcs-ese-2007.pdf>. 8
- [131] Danilo Sato, Alfredo Goldman, and Fabio Kon, *Tracking the evolution of object-oriented quality metrics on agile projects*, To be published in: 8th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2007), 2007, <http://www.dtsato.com/resources/default/xp-2007.pdf>. 8
- [132] Danilo Toshiaki Sato, *Uso eficaz de métricas em métodos Ágeis de desenvolvimento de software*, Master's thesis, Universidade de São Paulo, Departamento de Ciência da Computação, 2007. 119, 121
- [133] J.G. Schneider and L. Johnston, *eXtreme Programming at Universities—An Educational Perspective*, International Conference on Software Engineering, Portland, Oregon, USA (2003), 594–599. 9, 55
- [134] ———, *eXtreme Programming—helpful or harmful in educating undergraduates?*, The Journal of Systems & Software **74** (2005), no. 2, 121–132. 9, 55
- [135] P. Schuh, T.W. Inc, and IL Chicago, *Recovery, redemption, and extreme programming*, Software, IEEE **18** (2001), no. 6, 34–41. 9, 53
- [136] Ken Schwaber and Mike Beedle, *Agile software development with scrum*, Prentice Hall, 2001. 8, 39, 121
- [137] R. SCUPIN, *The KJ method: A technique for analyzing data derived from Japanese ethnology*, Human organization **56** (1997), no. 2, 233–237. 40
- [138] Sfetsos, Panagiotis, Angelis, Lefteris, Stamelos, and Ioannis, *Investigating the extreme programming system: An empirical study*, Empirical Software Engineering **11** (2006), no. 2, 269–301. 8
- [139] A. Shukla and L. Williams, *Adapting extreme programming for a core software engineering course*, Proceedings of the 15th Conference on Software Engineering Education and Training (CSEE&T 2002), 2002, pp. 184–191. 9, 55

- [140] Standish Group International, Inc., *Chaos report*, <http://www.standishgroup.com/press/article.php?id=2>, 2003. 2
- [141] J. Stapleton, *Dynamic systems development method: The method in practice*, Addison-Wesley, 1997. 8
- [142] M. Stephens and D. Rosenberg, *Extreme programming refactored: The case against xp*, Apress, 2003. 9
- [143] G. Succi, M. Stefanovic, and W. Pedrycz, *Quantitative assessment of extreme programming practices*, Canadian Conference on Electrical and Computer Engineering, 2001, vol. 1, 2001, pp. 81–86 vol.1. 8
- [144] S. Syed-Abdullah, M. Holcombe, and M. Gheorghe, *Practice Makes Perfect*, Proceedings of 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2003), vol. 2675, Springer, 2003, pp. 354–356. 9, 56
- [145] Frederick Winslow Taylor, *Scientific management*, Harper and Row, 1947. 4
- [146] Vinícius Manhães Teles, *Extreme Programming - Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*, Novatec, São Paulo, 2004. 8, 100
- [147] ———, *Um estudo de caso da adoção das práticas e valores do extreme programming*, Master's thesis, Universidade Federal do Rio de Janeiro, Departamento de Ciência da Computação, 2005. 9, 74, 107
- [148] B. Tessem, *Experiences in learning xp practices: A qualitative study*, Extreme Programming and Agile Processes in Software Engineering, 4th International Conference (Genova, Italy), Springer, 2003. 8
- [149] Alvin Toffler, *A terceira onda: a morte do industrialismo e o nascimento de uma nova civilização*, 26 ed., Record, Rio de Janeiro, 2001. 3
- [150] James E. Tomayko, *A comparison of pair programming to inspections for software defect reduction*, Computer Science Education **12** (2002), no. 3, 213–222. 9, 22
- [151] I. Tomek, *What i learned teaching XP*, Proceedings of the ACM OOPSLA Educators Symposium (Seattle, Washington, USA), November 2002, pp. 39–46. 9, 56
- [152] M. van Genuchten, *Why is software late? an empirical study of reasons for delay in software development*, IEEE Transactions in Software Engineering **17** (1991), no. 6. 2
- [153] VersionOne, *The state of agile development*, Available at <http://www.versionone.net/surveyresults.asp>, 2006. 8

- [154] Vinícius Manhães Teles, *Modelo de contrato de escopo negociável*, <http://www.improveit.com.br/blog/articles/2007/03/23/modelo-contrato-escopo-negociavel>, 2007, acessado em 26/04/2007. 49
- [155] William C. Wake, *Extreme Programming Explored*, Addison-Wesley, 2002. 8, 24
- [156] N. Wallace, P. Bailey, and N. Ashworth, *Managing XP with Multiple or Remote Customers*, Proceedings of the 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2002), 2002. 9, 40
- [157] L. Williams and R. Kessler, *Pair programming illuminated*, Addison-Wesley, 2002. 9
- [158] L. Williams, W. Krebs, L. Layman, and A. Antón, *Toward a framework for evaluating extreme programming*, Proceedings of the Eighth International Conference on Empirical Assessment in Software Engineering (EASE 04), 2004, pp. 11–20. 8
- [159] L. Williams, E. Wiebe, K. Yang, M. Ferzli, and C. Miller, *In support of pair programming in the introductory Computer Science course*, Computer Science Education **12** (2002), no. 3, 197–212. 9
- [160] L. A. Williams, *The collaborative software process*, Ph.D. thesis, University of Utah, Department of Computer Science, 2000. 9
- [161] L. A. Williams and R. R. Kessler, *All I really need to know about pair programming I learned in kindergarten*, Communications of the ACM **43** (2000), no. 5, 108–114. 9
- [162] L. A. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, *Strengthening the case for pair programming*, IEEE Software **17** (2000), no. 4, 19–25. 9
- [163] D. Wilson, *Teaching XP: a case study*, Proceedings of XP Universe 2001 (Raleigh, NC, USA), 2001. 9, 54
- [164] W. Wood and W. Kleb, *Exploring xp for scientific research*, IEEE Software **20** (2003), 30–36. 85
- [165] Klaus Wuestefeld, *Customer Proxy and Libero*, E-mail personal communication, September 2002. 40