Um Estudo Referente ao Processo de Documentação do Software Desenvolvido em Extreme Programming

Marcius da Silva da Fonseca

Um Estudo Referente ao Processo de Documentação do Software Desenvolvido em Extreme Programming

Marcius da Silva da Fonseca

UFSM

TRABALHO DE GRADUAÇÃO

UM ESTUDO REFERENTE AO PROCESSO DE DOCUMENTAÇÃO DO SOFTWARE DESENVOLVIDO EM EXTREME PROGRAMMING

Marcius da Silva da Fonseca

Curso de Ciência da Computação

Santa Maria, RS, Brasil
2007

UM ESTUDO REFERENTE AO PROCESSO DE DOCUMENTAÇÃO DO SOFTWARE DESENVOLVIDO EM EXTREME PROGRAMMING

por

Marcius da Silva da Fonseca

Monografia apresentada ao Curso de Ciência da Computação,
Área de Concentração em Engenharia de Software,
da Universidade Federal de Santa Maria (UFSM, RS),
como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientadora: Profª. Oni Reasilvia Sichonany

Trabalho de Graduação nº. 231

Santa Maria, RS, Brasil

2007

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

UM ESTUDO REFERENTE AO PROCESSO DE DOCUMENTAÇÃO DO SOFTWARE DESENVOLVIDO EM EXTREME PROGRAMMING

elaborado por
Marcius da Silva da Fonseca

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISÃO EXAMINADORA:

_____
Oni Reasilvia Sichonany, Me. (UFSM)
(Orientador)

_____
Raul Ceretta Nunes, Dr. (UFSM)

_____
Iria Brucker Roggia, **Me.** (UFSM)

Santa Maria, 01 de março de 2007.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

UM ESTUDO REFERENTE AO PROCESSO DE DOCUMENTAÇÃO DO SOFTWARE DESENVOLVIDO EM EXTREME PROGRAMMING

AUTOR: MARCIUS DA SILVA DA FONSECA
ORIENTADORA: ONI REASILVIA SICHONANY
Data e Local da Defesa: Santa Maria, 01 de Março de 2007.

A cada ano centenas de produtos de software são escritos no intuito de se resolver os mais diversos problemas e facilitar a execução de atividades cotidianas dos clientes, porém, estes muito freqüentemente não ficam satisfeitos.

Nas metodologias utilizadas até então, o processo de desenvolvimento de software era modelado de forma determinística, ou seja, era composto de várias fases bem definidas onde uma fase posterior era extremamente dependente dos resultados da fase anterior. Mudanças de projeto em fases iniciais resultavam em uma remodelagem de todo o processo, fazendo às vezes com que todo o projeto de software fosse reiniciado novamente.

O *Extreme Programming* é uma metodologia de desenvolvimento que se baseia em valores simples e algumas práticas que buscam oferecer flexibilidade e agilidade ao processo de software. Sua principal característica é a ênfase na codificação, ao invés dos conceitos de modelagem e *design* do software, e um dos seus princípios fundamentais é o teste do código, que é aplicado antes e durante a programação. Seu foco principal está no espaço de solução, procurando empregar ao máximo as técnicas de programação, para se obter a solução mais simples possível rapidamente.

Considerando a importância da documentação na manutenção do software e até mesmo para sua correta utilização, este trabalho se propõe a buscar e apresentar informações sobre os tipos de documentação sugeridos pela técnica XP.

Palavras-chave: Documentação de Software; Processo de Software; *Extreme Programming*.

ABSTRACT

Final Undergraduate Work
Computer Science
Universidade Federal de Santa Maria

A STUDY OF DOCUMENTATION PROCESS OF THE SOFTWARE DEVELOPED IN EXTREME PROGRAMMING

AUTHOR: MARCIUS DA SILVA DA FONSECA
ADVISOR: ONI REASILVIA SICHONANY
Date and Place of Presentation: Santa Maria, March 1st, 2007.

Every year, hundreds of software products are written aiming to solve several problems and to simplify the execution of customer quotidian tasks, however, they frequently doesn't feel satisfied.

On the methodologies utilized until now, the software development process was modeled on a deterministic way, being composed by innumerous well defined stages where a posterior phase was extremely dependent of the results from the anterior phase. Project changes in initial stages should result in a complete process redesign, causing sometimes a total restart of the entire software development process.

Extreme Programming is a new software development methodology based on simple practices that aim to offer flexibility and agility to the software process. Its principal features is coding emphasis instead of software design and modeling concepts. One of its fundamental principles is testing, applied before and during the coding. Its principal focus is on the solution space, trying to apply the programming techniques to their maximum, aiming to achieve the simplest possible solution rapidly.

Considering the importance of documentation to the software maintenance and correct utilization, this paper proposes to search and to introduce information about the documentation types offered by the XP technique.

Key-words: Software Documentation; Software Process; Extreme Programming

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Segundo Pfleeger [PFL2004], os projetos de software ocorrem de maneira semelhante ao processo de construção de uma casa. A cada ano milhares de casas são construídas e clientes satisfeitos se mudam para elas. Da mesma maneira, a cada ano centenas de produtos de software são construídos por desenvolvedores, porém, os clientes, muito freqüentemente, não ficam satisfeitos. Se é tão fácil diferenciar as etapas que compõem o processo de construção de um software, por que os engenheiros de software têm tanta dificuldade em construir produtos de software de qualidade?

No passado, assumia-se que desde o começo os clientes sabiam o que desejavam em seu software [PFL2004]. As técnicas de desenvolvimento amplamente utilizadas pregavam uma maior formalidade e preocupação com documentação, o que tornava o processo mais "pesado" e resistente às mudanças. Nesta metodologia, também referenciada por metodologia de desenvolvimento tradicional ou processo em cascata, o desenvolvimento de software era modelado de forma determinística. Buscava-se dividir o processo em inúmeras atividades especializadas, onde uma fase posterior era extremamente dependente dos resultados da fase anterior, de modo que o produto final fosse resultado da integração das inúmeras partes provenientes destas atividades ao longo do processo [TEL2004].

As pesquisas e a educação no campo da Engenharia de Software, durante os últimos 30 anos, enfatizaram controle sobre esse tipo de processo de desenvolvimento: a comunicação entre os desenvolvedores e os clientes era terminada assim que as especificações dos requerimentos eram levantadas, baseando-se na premissa de que o controle detalhado sobre as diversas fases que compõem o processo de desenvolvimento de software bastaria para garantir a qualidade do software [HAM2006].

Porém, essa limitação na comunicação entre desenvolvedores e clientes geralmente resultava no enrijecimento do processo, tornando-o suscetível a falhas e resistente a mudanças. Alterações de projeto em fases iniciais resultavam em uma remodelagem de todo o processo, fazendo às vezes que todo o projeto de software fosse reiniciado novamente.

Devido a essa falta de flexibilidade, somado ao atual cenário corporativo, que é caracterizado por pressões e necessidades de se obter respostas rápidas às mudanças do mundo dos negócios as quais devem ser incorporadas no conteúdo dos produtos [KET2005], e o rápido avanço tecnológico apresentado nos últimos anos, principalmente do fim da década de 90 em diante [AGI2006], vêm se tornando cada vez mais populares as metodologias

conhecidas como Processos Ágeis de Desenvolvimento, dentro da qual o *Extreme Programming* (XP) se enquadra.

O XP é uma metodologia de desenvolvimento que se baseia em valores simples e algumas práticas que buscam oferecer flexibilidade e agilidade ao processo de software. Suas principais características são: implementação intensa, freqüente comunicação com o cliente, *releases* interativos (incremental) e a baixa preocupação com formalizações e documentação.

No entanto, considerando que os avanços na área da Ciência da Computação se devem em parte aos níveis de abstração alcançados ao longo dos tempos e que a falta de documentação ou a baixa qualidade da mesma pode dificultar a tomada de decisão ou a manutenção do processo, gerando um fenômeno conhecido por degeneração arquitetural [HOC2005], este projeto se propõe a buscar e apresentar informações sobre os tipos de documentação sugeridos pela técnica e quais são os esforços que vêm sendo feitos na busca pelo aperfeiçoamento do processo de modelagem e documentação utilizado por equipes de desenvolvimento XP.

Para se atingir o objetivo proposto, uma revisão de literatura a cerca das metodologias estudadas é apresentada no capítulo 2.

No capítulo 3 é apresentada a metodologia de documentação adotada pelo processo XP, assim como uma descrição detalhada de cada um dos artefatos pesquisados que compõem o conjunto de elementos de documentação do processo XP. Neste capítulo também é apresentada a metodologia *Agile Modeling* (AM) – que visa aprimorar a maneira como é feita a documentação tanto em processos ágeis como tradicionais de desenvolvimento – assim como uma seção de discussões sobre os aspectos envolvidos nesse processo de documentação.

No capítulo 4 estão mostrados alguns aspectos referentes a experiências de aplicação dos conceitos do XP e sua documentação, assim como uma proposta para um trabalho futuro. Por fim, no capítulo 5 estão apresentadas as conclusões finais do trabalho.

2 AS METODOLOIAS DE DESENVOLVIMENTO DE SOFTWARE

No decorrer dos seus anos de existência, a Engenharia de Software tem trilhado seu caminho na busca por metodologias e técnicas que ajudem os profissionais de Sistemas da Informação a produzirem software de maior qualidade e robustez, além da redução de custos e a satisfação dos clientes. Este processo de busca por melhores soluções inspirou-se inicialmente nos métodos de produção industrial e vem evoluindo até os dias de hoje.

Este capítulo possui o objetivo de apresentar essa inspiração inicial que resultou nas metodologias tradicionais. Nas seções a seguir, também será apresentada uma visão geral dos aspectos mais importantes referente às metodologias tradicional e XP de desenvolvimento de software, assim como as principais vantagens e limitações impostas pelo XP.

2.1 A evolução dos métodos de produção

Em um processo de fabricação industrial, diversas atividades são desempenhadas de modo que o conjunto de matérias-primas, que alimenta este processo de fabricação, se transforme no produto desejado, definindo-se assim uma linha de montagem. Neste processo, as transformações pelas quais as matérias-primas passam são determinísticas, ou seja, a aplicação destas transformações gera um resultado previamente conhecido. Tal metodologia é fruto de estudos e constatações de Frederick Winslow Taylor, publicados no início do século XX em um dos seus mais importantes artigos, referenciado por "*The Principles of Scientific Management*", em que procurava estudar e implementar metodologias de gerenciamento de produção baseadas em conceitos cientificamente testados [TAY1911].

Taylor acreditava que o gerenciamento do processo produtivo contemporâneo era muito superficial e defendia o seu estudo como uma disciplina, onde os trabalhadores deveriam cooperar com o gerenciamento, no qual os melhores resultados seriam obtidos a partir da parceria entre um gerenciamento treinado e qualificado e uma força de trabalho cooperativa e inovadora, com cada uma das partes necessitando da outra. Deste modo, Taylor desenvolveu os cinco princípios do gerenciamento científico:

- Estudar especificamente cada uma das partes da tarefa e desenvolver a melhor maneira para executá-la;
- Selecionar a melhor pessoa para fazer o trabalho;
- Treinar, ensinar e desenvolver o trabalhador;

- Prover incentivos financeiros para métodos seguintes;
- Dividir trabalho e responsabilidade, fazendo com que os gerenciadores sejam responsáveis pelo planejamento e os trabalhadores sejam responsáveis pela execução do trabalho, respectivamente.

Com o passar dos anos, essas constatações acerca das teorias de produção estudadas por Taylor ficou sendo conhecida por Taylorismo.

A partir da Revolução Industrial, as constatações de Taylor passaram a ser amplamente utilizadas [TOF1980], tendo contribuído enormemente para a evolução da indústria até chegarmos aos métodos de produção que conhecemos hoje.

Analisando as informações expostas anteriormente e observando o modo como ocorre o desenvolvimento de software atualmente, fica evidente a influência das recomendações de Taylor na área da produção e engenharia de software.

2.2 Uma visão geral da metodologia tradicional de desenvolvimento

A metodologia de desenvolvimento tradicional[1] segue um planejamento seqüencial, na qual o processo de software é dividido em partes especializadas, onde o desenvolvimento do software em questão ocorre seguindo a execução destas tarefas especializadas, de forma que o resultado obtido em uma dada fase segue como entrada para a fase seguinte.

O ciclo de vida de um software concebido através da aplicação do modelo em cascata é constituído pelos estágios, esquematizados na figura 2.1.

Este modelo de desenvolvimento de software é o modelo mais antigo e vem sendo utilizado desde meados da década de 70.

[1] 1 Nota: entende-se por desenvolvimento tradicional as metodologias baseadas no modelo em cascata, que já fora descrito anteriormente.

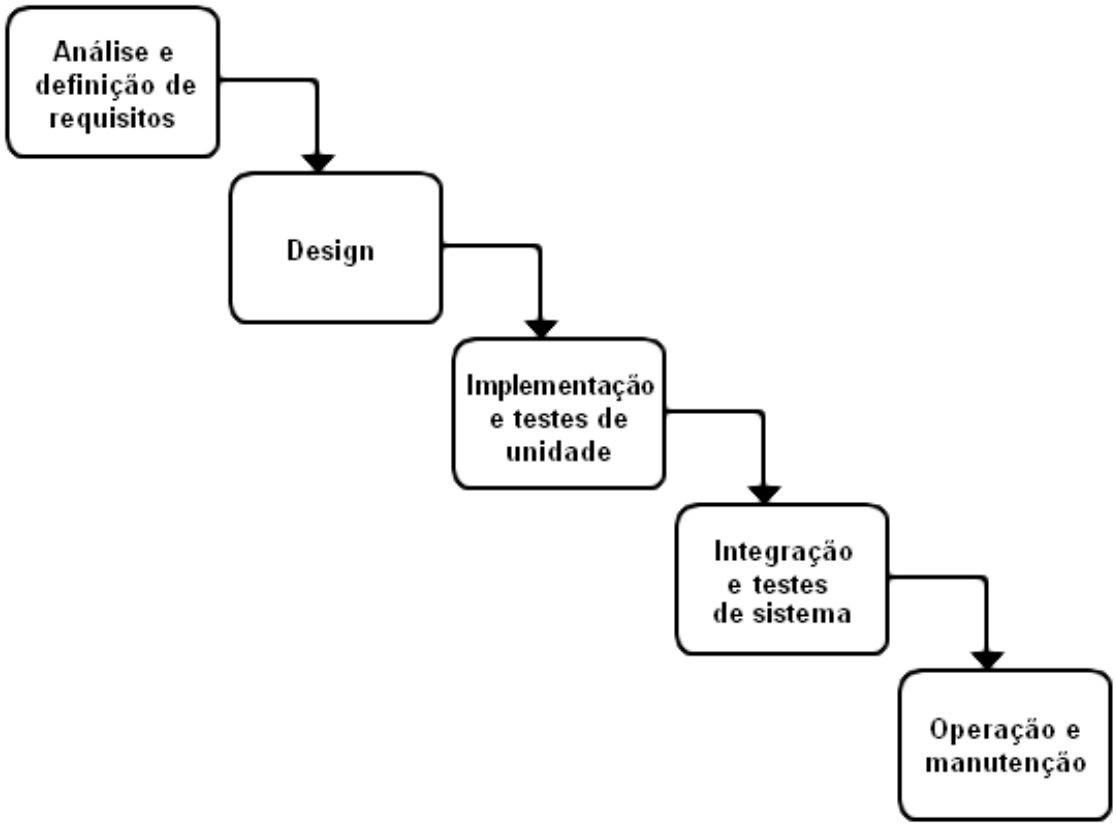

Figura 2.1 – O ciclo de vida do modelo em cascata [SOM2003].

Neste tipo de processo podemos observar uma grande dificuldade de se acomodar mudanças após o início do projeto, devido à grande quantidade de atividades e artefatos que procuram proteger o software contra modificações. Para que se possa avançar para a fase subseqüente uma fase deve ser completada.

Além disso, temos que considerar as seguintes questões:

- O particionamento do projeto nas fases distintas torna difícil de responder às mudanças dos requisitos do usuário, deste modo este modelo só será apropriado caso os requisitos do projeto sejam bem conhecidos e com chances muito limitadas de mudança. No entanto, projetos reais raramente seguem o fluxo seqüencial;
- Dificilmente o cliente conseguirá definir todas as exigências explicitamente;
- Apenas após um ponto tardio do cronograma do projeto haverá uma versão do trabalho disponível para análise.

2.3 Uma visão geral da metodologia XP

O XP é uma metodologia integrante da categoria chamada de Processos Ágeis de Desenvolvimento, concebida para atender as necessidades específicas do desenvolvimento de software conduzido por equipes de pequeno e médio porte que enfrentam situações de requerimentos vagos, os quais mudam freqüentemente durante o processo [BEC2000a]. É uma metodologia leve, centrada no conjunto de valores práticas de desenvolvimento [BEC2000a, TEL2004, SCH2005], que promove o processo de desenvolvimento de forma interativa, cooperativa e adaptativa que começa com um *design* simples que vai ao encontro de um conjunto de requerimentos inicial [SCH2005]. Enfatiza iterações individuais ao invés de processos e ferramentas, implementação intensa ao invés de documentações, colaboração dos clientes ao invés de negociação de contrato e respostas às mudanças ao invés de seguir um plano rígido [SAK2005].

Vale aqui ressaltar que embora a metodologia adote um enfoque maior na codificação ao invés da documentação, isto não significa que o XP desconsidere a necessidade de se documentar o software produzido, como será detalhado posteriormente.

Esse novo paradigma referenciado por “desenvolvimento ágil” sugere que todas as fases do desenvolvimento tradicional (em cascata) sejam realizadas diversas vezes no decorrer do projeto, como é mostrado na figura 3.1, compondo assim um conjunto de ciclos que se repetem. Cada um destes ciclos compreende uma iteração [TEL2004].

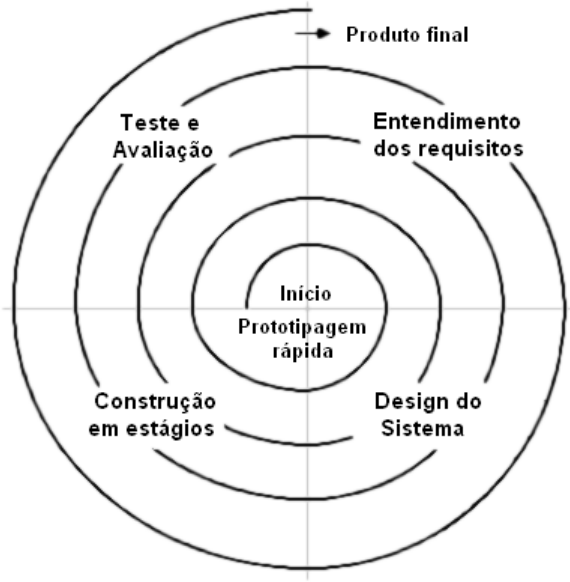

Figura 2.2 – O ciclo de vida do modelo iterativo [BEC2000a].

Deste modo, à medida que as iterações são concluídas temos um protótipo funcional testado e pronto para uso, sendo que com o evoluir do processo novas versões serão produzidas, englobando um número cada vez maior de funcionalidades até se chegar ao produto final que conterá todas as funcionalidades requisitadas pelo cliente.

Como citado anteriormente, a metodologia XP é centrada em um conjunto de valores e práticas de desenvolvimento. Este conjunto é organizado de forma harmônica e coesa, visando assegurar que o cliente receba um retorno de qualidade em relação ao seu investimento de software.

Os valores fundamentais do XP:

- Comunicação;
- Simplicidade;
- *Feedback*;
- Coragem.

As práticas do XP:

- Cliente presente;
- Jogo do planejamento;
- *Stand up meeting*;
- Programação em par;
- Desenvolvimento guiado pelos testes;
- *Refactoring*;
- Código coletivo;
- Código padronizado;
- *Design* Simples;
- Metáfora;
- Ritmo sustentável;
- Integração contínua;
- *Releases* curtos.

A seguir, na figura 3.2 temos um diagrama que apresenta a relação entre as práticas que compõem o XP:

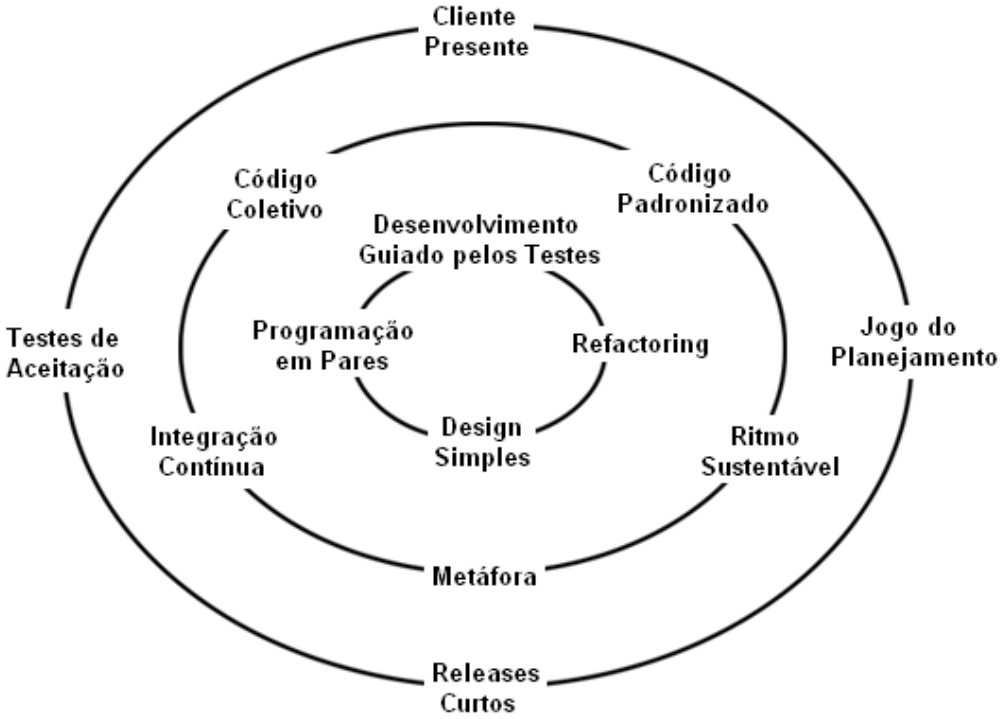

Figura 2.3 – Relação entre as práticas do XP [TEL2004].

Nos próximos itens deste capítulo será apresentado um detalhamento a cerca de cada um dos valores e práticas adotadas pela metodologia.

2.3.1 Os valores fundamentais

- Comunicação:

Um programador pode revelar notícias desfavoráveis ao gerenciador, e este acaba por punir o programador. Ou ainda, o cliente fornece ao programador alguma informação importante, e o programador esquece ou não dá a atenção necessária a esta informação. Estes são alguns exemplos citados por [BEC2000a] que podem levar a uma má comunicação em um projeto. Por este motivo, a comunicação é um valor importante no XP para que os detalhes do processo de desenvolvimento sejam tratados com a devida atenção.

- Simplicidade:

Para que o cliente possa gerar *feedback* rapidamente é necessário simplicidade na comunicação com o cliente. Como se pode ver, simplicidade e comunicação são valores que estão altamente correlacionados. Quanto mais simples for seu sistema, menos você precisará comunicar em relação a ele, alcançando-se deste modo uma comunicação mais completa, especialmente se for possível simplificar o sistema o suficiente a ponto de se requerer menos programadores para implementá-lo [BEC2000a].

- Feedback:

O *feedback* funciona em conjunto com a comunicação e a simplicidade. Quanto mais *feedback* é adquirido, mais fácil se torna a comunicação [BEC2000a]. Deste modo, quando o cliente aprende com o sistema e reavalia suas necessidades ele gera *feedback* para a equipe de desenvolvimento [TEL2004]. O *feedback* é um mecanismo fundamental para que o cliente conduza o desenvolvimento do software para aquilo que irá gerar mais valor.

- Coragem:

Dado que o sistema é implementado de forma incremental, a equipe de desenvolvedores estará fazendo a manutenção do software e implementando novas funcionalidades. Em muitos casos, poderá acontecer de que algo que já vinha funcionando corretamente seja alterado, tendo assim o risco de se gerarem falhas no sistema. Por esta razão, a equipe precisa ser corajosa e acreditar que será capaz de fazer o software evoluir com segurança e agilidade [TEL2004].

2.3.2 As práticas do XP

- Cliente presente:

O XP sugere a presença do cliente em tempo integral no dia-a-dia do projeto sempre que possível como forma de se alcançar o sucesso do mesmo. Essa participação é importante, pois é através dela que a simplicidade do processo é alcançada, além de permitir que o

desenvolvimento do software evolua através de pequenos ajustes ao invés de mudanças bruscas ao longo do tempo.

- Jogo do planejamento:

O planejamento é peça fundamental para assegurar que o esforço desprendido em cada dia do desenvolvimento irá gerar o valor solicitado pelo cliente. Este planejamento trata de diversas questões do projeto, entre os quais podemos citar:

- Divisão de responsabilidades entre cliente e equipe;
- Gerenciamento das estórias;
- Estimação do custo de desenvolvimento de cada estória;
- Planejamento e encerramento dos *releases* e iterações.

Este jogo do planejamento é feito diversas vezes no decorrer do projeto, no início de cada iteração.

- Stand up meeting:

Como o próprio nome sugere (*stand up meeting* significa “reunião em pé” em inglês), trata-se de uma curta reunião que é feita no início de cada dia de trabalho, onde os desenvolvedores expõem o trabalho executado no dia anterior, discutem eventuais problemas, definem as tarefas que deverão ser desenvolvidas e quem irá desenvolver cada uma delas, além de também definirem as prioridades, fazendo com que os esforços de trabalho para o dia que se inicia sejam direcionados da maneira mais adequada. Também é importante citar que é através do *stand up meeting* que os desenvolvedores adquirem uma visão global do estado de desenvolvimento do projeto.

- Programação em par:

A programação em par é uma técnica na qual dois programadores trabalham em um mesmo problema, ao mesmo tempo e em um mesmo computador. Enquanto um dos desenvolvedores, no papel de condutor, assume o teclado e escreve o código do programa, o outro no papel de navegador o acompanha fazendo o trabalho estratégico.

Como vantagem da prática da programação em pares, podemos citar a facilitação da tarefa de revisão e correção do código, favorecimento à modelagem de soluções simples e inovadoras através da soma de idéias e um maior suporte à disseminação do conhecimento entre os desenvolvedores envolvidos.

- Desenvolvimento guiado pelos testes:

Antes de começarem a escrever o código, os desenvolvedores escrevem testes de unidade, visando aperfeiçoar seu conhecimento sobre as necessidades do cliente. Esta preocupação com os testes acaba por definir as interfaces (assinaturas) dos métodos e classes, proporcionando assim um melhor design. Deste modo, os desenvolvedores possuem em mente exatamente o que é preciso codificar para implementar as funcionalidades requeridas, além de construir um conjunto amplo de testes que pode ser usado sempre que necessário para validar todo ou uma parte do sistema em desenvolvimento.

- Refactoring:

A prática de *refactoring* consiste na modificação/aperfeiçoamento do código produzido pela equipe. Ela é feita sempre que seja constatada a necessidade de uma modificação que torne o código mais simples, legível e limpo.

No XP o *refactoring* é obrigação de todos os desenvolvedores sempre que necessário, independendo de qual parte do código esteja em questão. Este investimento constante de atenção na legibilidade e simplicidade do código, reescrevendo-se suas funcionalidades sempre que necessário proporciona um código muito mais organizado, facilitando a implementação de eventuais alterações mais complicadas sem que ocorra um consumo excessivo de tempo para tal.

- Código coletivo:

No XP, toda e qualquer parte do código-fonte pode ser acessado e alterado por qualquer membro da equipe de desenvolvimento, sem a necessidade de se obter algum método de autorização. Esta prática tem por objetivo proporcionar um mecanismo adicional de revisão e verificação de código, além de proporcionar uma maior legibilidade do mesmo,

pois se algum desenvolvedor que obteve acesso em uma determinada faixa de código observou que o mesmo está confuso, este desenvolvedor poderá fazer o *refactoring* para tornar o código mais legível.

- Código padronizado:

Devido ao compartilhamento de acesso ao código-fonte, para que haja uma rápida e eficaz manipulação do mesmo por parte dos diversos desenvolvedores da equipe, é necessário que a equipe adote uma série de critérios e convenções que definam um padrão de codificação em comum.

- Design Simples:

De modo a se obter agilidade no processo e um rápido *feedback* por parte do cliente, o XP faz uso da simplicidade no processo de design do sistema, evitando a criação de generalizações que tenham em mente a previsão de necessidades futuras. O design simples provê apenas um planejamento das funcionalidades requeridas pelo cliente naquele momento, partindo da premissa de que o processo suportará a incorporação de novas funcionalidades e alterações, com base nas práticas de *refactoring* e dos sucessivos testes de unidade a que o sistema é submetido.

- Metáfora:

Esta pratica tem por finalidade proporcionar uma facilidade de comunicação entre o cliente e a equipe de desenvolvimento, visto que a metáfora tem o poder de transmitir idéias complexas de maneira simples e objetiva. Isto influencia na agilidade com que o *feedback* será gerado, e conseqüentemente na simplicidade do design do sistema a ser desenvolvido.

- Ritmo sustentável:

Para garantir que a equipe de desenvolvimento mantenha o rendimento adequado para a produção de software de qualidade, o XP adota uma filosofia na qual os desenvolvedores

não devem trabalhar mais do que oito horas diárias, visto que uma mente descansada é requisito básico para o estabelecimento de um trabalho eficiente ao longo do dia.

- Integração contínua:

Sempre que uma nova funcionalidade é implementada e anexada ao software, deve-se executar o processo de integração desta nova porção de código. Também se devem efetuar os testes de unidade, visto que uma integração pode introduzir erros no sistema.

Através desta integração contínua, possíveis falhas serão abordadas rapidamente, evitando que estas se tornem problemas maiores no futuro.

- Releases curtos:

Um *release* consiste em um conjunto de funcionalidades implementadas que representam um valor bem definido para o cliente. O XP busca subdividir o desenvolvimento em *releases* de modo que estes sejam tão curtos quanto possível.

2.4 Os benefícios e limitações do XP

O XP, assim como qualquer teoria na área da computação, tem sua aplicação recomendada em determinadas situações, porém também há ocasiões em que ela não é a melhor solução na construção de um software. Por vezes ela pode até mesmo ser inviável devido às circunstâncias e a natureza do projeto em questão.

Antes de se tomar uma decisão definitiva, quanto à metodologia de desenvolvimento a ser assumida em um projeto, é preciso ter o conhecimento pleno das metodologias disponíveis e as informações relevantes do projeto a fim de se discernir àquelas metodologias que podem atender satisfatoriamente aos requisitos do referido projeto.

Com base nos estudos de KETTUNEN e LAANTI [KET2005], que buscaram em suas pesquisas elaborar um plano de auxílio à seleção do modelo de processo a ser aplicado em um projeto de software, podemos expor algumas considerações a cerca do XP, denotando suas vantagens em relação a certas situações, algumas observações de situações onde é preciso muita cautela para que o processo evolua de forma satisfatória, bem como informações que indicam as situações onde o XP não é viável ou aconselhável.

2.4.1 Os benefícios do XP

A forte ligação entre a equipe e o cliente, juntamente com sessões de planejamento semanais (no mínimo), faz com que os objetivos não muito bem definidos no início do projeto não sejam um problema no XP. Mudanças de requerimentos também são bem vindas, visto que a natureza do XP provê uma oportunidade de redefinição com base semanal, porém, a metodologia não oferece um gerenciamento formal das mudanças ao longo do processo.

Essa comunicação intensa e direta com o cliente durante o decorrer de todo o processo também tem importância no sentido de que o software terá exatamente as funcionalidades pretendidas pelo mesmo. É o cliente que decide quais funcionalidades serão implementadas para o próximo release, durante o jogo do planejamento. Nenhuma funcionalidade será acrescida ao projeto sem o consentimento do cliente.

A ênfase ao planejamento, assim como seu constante ajuste para que os planos se adaptem à realidade do projeto, garante ao processo que o esforço desprendido no ato do planejamento não seja nem excessivo nem faltoso.

O XP não é recomendado para projetos muito grandes, no entanto, o XP pode ser aplicado a tal tipo de projeto desde que o mesmo possa ser dividido em vários subprojetos menores, correndo em paralelo, cada um tratando de suas próprias estórias. O XP trabalha com equipes de desenvolvimento, com no máximo até 12 desenvolvedores.

Na engenharia de software, projetos que excedam 50% ou mais do tempo destinado para sua conclusão são considerados projetos “a caminho da morte”. No XP, isto não ocorre, pois o cliente possui um contato muito próximo da equipe durante todo o tempo, sendo assim muito visível o progresso feito no desenvolvimento. Isto é o suficiente para fazer com que o cliente espere o tempo necessário para que tenha o produto desejado. Também vale lembrar que o projeto pode ser cancelado sem o problema de o cliente receber apenas modelos e documentos, ou seja, ao final de um projeto XP o usuário sempre irá receber algo útil.

2.4.2 As limitações do XP

Pela sua definição, a equipe do projeto deve estar em um local definido e centralizado. Além disso, há a necessidade de se manter o cliente ou um representante do mesmo no local do projeto. Caso o cliente não esteja tencionado a seguir essa prática, o XP não deverá ser aplicado.

O XP trabalha com um número máximo aproximado de dez desenvolvedores. Deste modo, projetos de grande proporção não devem ser abordados utilizando XP, no entanto, se o projeto for factível de subdivisão, é possível ter-se múltiplas equipes XP trabalhando concorrentemente, cada uma trabalhando com suas próprias estórias. Atenta-se para o fato de que pode não ser viável a prática de se gerar novos releases para o cliente em um cronograma semanal em projetos de sistemas maiores.

A metodologia não cobre qualquer detalhe em relação ao conjunto de ferramentas que devem ser utilizadas, porém não é uma boa idéia possuir uma equipe de desenvolvedores altamente experientes e equipá-los com ferramentas limitadas. No entanto, deve-se observar que o XP não deve ser utilizado em conjunto com ferramentas totalmente novas e desconhecidas pelos desenvolvedores.

2.4.3 Situações de risco

As situações apresentadas a seguir não inviabilizam imediatamente a aplicação do processo XP, porém devem ser analisadas com muita atenção para que o processo evolua com segurança.

O XP é otimizado para desenvolvimentos rápidos. Porém, alguns custos devem ser levados em consideração e balanceados ao se começar um projeto em XP, tais como o nível de experiência requerida da equipe e o tempo disponível que o cliente possui para manter a proximidade necessária no projeto. Tanto o cliente e a equipe devem concordar com a agenda do projeto.

No XP, pequenas alterações podem ser implementadas, porém, deve se ter em mente que o XP não provê um suporte robusto ao design da arquitetura (apenas metáforas e design simples). Deste modo, alterações mal planejadas podem resultar na degeneração da arquitetura do projeto.

Outro ponto que se deve tomar cuidado é quanto à qualidade da comunicação entre os membros da equipe, incluindo o cliente. O XP se baseia na comunicação freqüente e aberta.

Falhas nesta comunicação podem causar sérios danos à continuidade do processo, podendo até mesmo ser fatal para o projeto, visto que os problemas de comunicação podem resultar em uma perda da conexão entre os desenvolvedores e o cliente.

O XP tem como característica a rápida e visível evolução do processo de software, deste modo, dependências externas (tais como subcontratos) lentas e/ou imperfeitas podem fazer com que se tenha toda a equipe esperando por determinado recurso.

No XP, não há muito espaço para a inclusão de desenvolvedores novatos (estagiários). A perda de pessoas-chave do projeto pode ser resolvida através da prática do replanejamento (jogo do planejamento), no entanto, mudanças súbitas de equipe podem representar um sério problema, visto que a principal fonte tangível de informação é o código-fonte produzido (a documentação é simples).

3 A DOCUMENTAÇÃO DE SOFTWARE

Nas próximas seções serão apresentadas as questões relevantes referentes à modelagem e documentação de software, tais como a importância da documentação para a manutenção de software em geral e como o XP enquadra a prática da documentação em seus projetos.

3.1 A importância da documentação:

A documentação de software é o importante conjunto de artefatos que nos conta toda a história de um sistema, desde sua concepção, passando pelo seu desenvolvimento e acompanhando sua manutenção, nos mostrando o que o software faz, como faz e para quem faz [HOF2002].

Sua importância reside no fato de que é através dela que a evolução do software é registrada para que sejam criadas as bases necessárias para as etapas posteriores do processo de software, incluindo treinamento e utilização do mesmo. É, também, através dessa documentação que os desenvolvedores capturam as informações e conhecimento necessário acerca de um software, de modo a tornar tangível a sua manutenção.

Outro ponto importante é que a modelagem e a documentação oferecem suporte para uma comunicação mais concisa e eficiente dentro da equipe de desenvolvimento, como será visto na seção 3.3, pois esses elementos promovem um melhor planejamento das ações e transformam situações complexas em diagramas e documentos de fácil entendimento, resultando assim na diminuição da taxa de erros e, conseqüentemente, na redução dos custos de desenvolvimento.

Assim que o software produzido é liberado para uso operacional, não significa que o mesmo não mais necessite intervenções de desenvolvedores, sejam estes participantes ou não do processo de desenvolvimento do software em questão. Dá-se então, início à etapa de manutenção, considerada por muitos profissionais da área da Engenharia de Software como sendo uma das fases mais problemáticas e dispendiosas dentro do ciclo de vida de um software, visto que intervenções em um software finalizado oferecem maior complexidade e menor flexibilidade [COS1996].

De acordo com o motivo das alterações necessárias, a manutenção de software pode ser classificada em quatro categorias [PRE1992]:

- Manutenção Corretiva, que tem por objetivo corrigir erros não detectados durante o desenvolvimento.
- Manutenção Adaptativa, que visa adequar o software a alterações de hardware ou de ambiente do usuário.
- Manutenção Evolutiva, que procura atender a novas solicitações do usuário, tais como inclusão de novas capacidades, melhorias de desempenho e funcionalidades.
- Manutenção Preventiva, que tem por objetivo aumentar a facilidade de manutenção, procurando tornar o software mais compreensível e alterável.

Para que o processo de manutenção de um determinado software seja concluído com êxito, é necessário que o desenvolvedor que fará a manutenção seja capaz de compreender o código para que as alterações ocorram de forma rápida e segura.

Durante as últimas décadas, desde o surgimento da Engenharia de Software, muitos estudiosos e profissionais da área procuraram aprimorar as metodologias de processos de software. Nesse período, foram criados e formalizados diversos tipos de artefatos de documentação com o intuito de se obter níveis de abstração mais altos e assim facilitar a evolução e gerenciamento dos processos de software.

Devido à atual variedade de tipos de artefatos de documentação existentes, a escolha de um conjunto ideal de artefatos que comporte as informações essenciais acerca de um sistema pode se tornar uma tarefa complicada.

Tendo este cenário em foco, SOUSA *et. al* [SOU2004a, SOU2004b] realizaram dois estudos cujo objetivo era facilitar a tomada de decisão referente a escolha dos artefatos, os quais devem ser ostentados como componentes do conjunto de documentação essencial de um processo de software.

Em um primeiro estudo foram analisados quais são os principais instrumentos de documentação de software. Esta análise consistiu na apresentação de um conjunto extenso de tipos de artefatos na forma de um questionário para que profissionais experientes da área de manutenção de software indicassem o grau de importância referente a cada um dos itens apresentados. Tais opiniões foram então contabilizadas, resultando os dados apresentados na tabela 3.1.

Tabela 3.1. Importância dos artefatos da orientação a objetos [SOU2004a]				
Artefatos	**Sem Importância**	**Pouco Importante**	**Importante**	**Muito Importante**
Código Fonte	0,0%	0,0%	6,2%	93,8%
Comentários do Código Fonte	0,0%	7,6%	15,2%	77,3%
Modelo Lógico de Dados	0,0%	4,7%	23,4%	71,9%
Diagrama de Classes	0,0%	5,6%	31,5%	63,0%
Modelo Físico de Dados	0,0%	3,1%	37,5%	59,4%
Diagrama de Caso de Uso	0,0%	10,7%	35,7%	53,6%
Caso de Uso Especificado	1,9%	7,7%	38,5%	51,9%
Dicionário de Dados	1,6%	17,5%	33,3%	47,6%
Modelo Conceitual de Dados	6,5%	12,9%	35,5%	45,2%
Plano de Testes de Homologação	9,8%	14,8%	31,1%	44,3%
Manual do Usuário	9,5%	17,5%	34,9%	38,1%
Plano de Testes de Sistema	6,3%	17,5%	39,7%	36,5%
Plano de Implantação	7,8%	15,6%	42,2%	34,4%
Plano de Testes Unitário	8,1%	21,0%	37,1%	33,9%
Diagrama de Seqüência	0,0%	10,9%	56,4%	32,7%
Protótipo Funcional	11,5%	21,3%	36,1%	31,1%
Diagrama de Atividades	5,6%	13,0%	51,9%	29,6%
Documento de Visão	4,1%	24,5%	42,9%	28,6%
Plano de Migração de Dados	11,7%	18,3%	41,7%	28,3%
Protótipo não Funcional	13,8%	22,4%	43,1%	20,7%
Glossário	7,9%	33,3%	38,1%	20,6%
Diagrama de Componentes	9,6%	23,1%	48,1%	19,2%
Diagrama de Estados	11,8%	29,4%	47,1%	11,8%
Diagrama de Colaboração	12,5%	35,4%	45,8%	6,3%
Diagrama de Pacotes	9,8%	33,3%	51,0%	5,9%

No estudo seguinte, foram avaliadas a disponibilidade e o uso de cada um dos itens de documentação durante o ciclo de manutenção de acordo com a experiência dos profissionais, porém, nesta fase os dados apresentados são referentes à análise estruturada. Na tabela 3.2 temos a apresentação dos dados resultantes desta pesquisa.

Tabela 3.2. Utilização e disponibilidade dos artefatos de documentação [SOU2004b]		
Artefatos	**Utilização**	**Disponibilidade**
Código Fonte	100,0%	100,0%
Comentários do Código Fonte	100,0%	100,0%
Modelo Lógico de Dados	100,0%	85,7%
Descrição dos Requisitos	40,0%	71,4%
Modelo Físico de Dados	85,7%	100,0%
Lista de Requisitos	40,0%	71,4%
Dicionário de Dados	85,7%	100,0%
Modelo Conceitual de Dados	40,0%	71,4%
Plano de Testes de Homologação	0,0%	42,9%
Manual do Usuário	16,7%	85,7%
Plano de Testes de Sistema	42,9%	100,0%
Plano de Implantação	33,3%	85,7%
Plano de Testes Unitário	50,0%	85,7%
Protótipo Funcional	20,0%	71,4%
Diagrama de Fluxo de Dados	0,0%	42,9%
Modelo de Arquitetura	0,0%	71,4%
Especificação de Componentes	0,0%	57,1%
Plano de Migração de Dados	0,0%	57,1%
Diagrama Hierárquico de Funções	0,0%	42,9%
Diagrama de Contexto	0,0%	57,1%
Funções Derivadas dos Requisitos	0,0%	57,1%
Protótipo não Funcional	0,0%	57,1%
Glossário	20,0%	71,4%
Diagrama Geral de Transações	0,0%	57,1%

Os dados apresentados na tabela 3.1 mostram que a documentação possui um papel fundamental no que tange a manutenção de software. Esta constatação fica evidenciada pelo fato de que os artefatos que obtiveram um menor índice de importância tiveram mais de 50% de indicação dos mantenedores.

Deste modo, podemos inferir que a capacidade e a facilidade que um software possui de receber manutenção estão primariamente relacionadas à documentação disponível e sua qualidade.

3.2 A documentação no XP

Na literatura, o XP geralmente é referenciado como sendo uma metodologia leve de desenvolvimento que visa o espaço de solução, buscando maximizar o uso de técnicas de programação para obter a solução mais simples o mais rápido possível [INS2002, SOU2004a]. Muito pouco se fala da documentação presente no processo XP, fazendo com que a maioria das pessoas acredite que não exista documentação nos projetos que o utilizam e que

a velocidade do processo venha exatamente desta inexistência da necessidade de se documentar.

Porém, ao contrário do que foi dito anteriormente, existe uma preocupação com o processo de documentação do software produzido no XP. No entanto, este processo de documentação ocorre de uma maneira diferente à dos processos tradicionais.

A maioria dos projetos de desenvolvimento se baseia em processos prescritivos que evoluem através da geração de inúmeros documentos, passados de etapa em etapa durante o desenvolvimento, ou seja, cada uma das etapas do processo de software tradicional – enumeradas na figura 2.1 – gera um conjunto de artefatos de modelagem e documentação que servirão de base ao estágio posterior, resultando assim em um conjunto muito volumoso de documentos para se gerenciar.

Esta prática de documentação excessiva é resultante da influência do enunciado de Barry Boehm [TEL2004], na qual se afirmava que o custo de alteração de um software aumenta exponencialmente no decorrer do ciclo de vida do software, como é apresentada na figura 3.1, ou seja, uma alteração de software representa um alto custo e precisa ser evitado. Com base neste fato, admitia-se que era mais fácil manipular documentos em vez de se efetuar alterações no código.

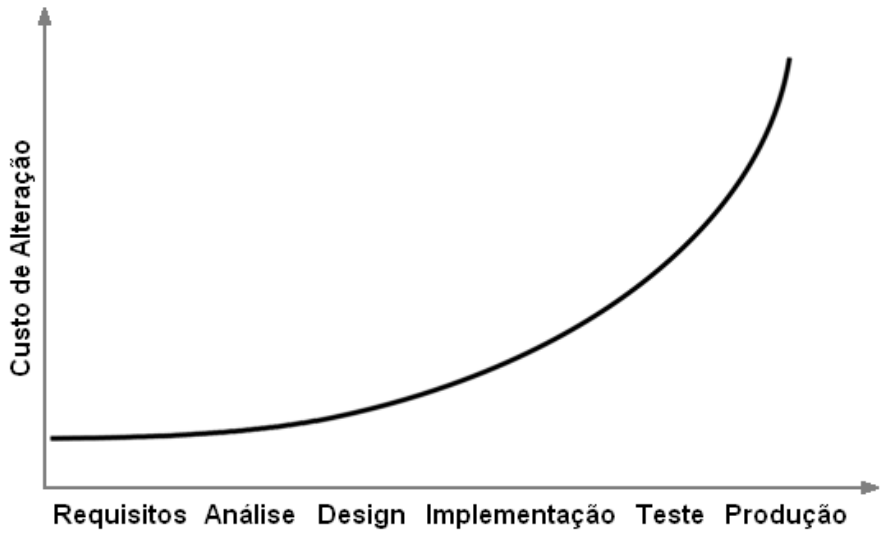

Figura 3.1 – Curva de custo de alteração de software segundo Boehm [BEC2000a].

Essa abordagem que enfatiza a documentação tem por objetivo minimizar a probabilidade de se cometer erros durante o processo de desenvolvimento do software, porém, ela também pode ser responsável por inúmeros efeitos colaterais.

Um desses efeitos colaterais que podemos observar é a perda de flexibilidade do projeto, ou seja, uma vez finalizada a fase de levantamento de requisitos, dificilmente haverá espaços para modificações. Também podemos citar a diminuição no ritmo de desenvolvimento, uma vez que além de se produzir código também se tem a preocupação em relação ao gerenciamento de uma vasta gama de documentos e modelos, visto que estes documentos devem estar sempre atualizados e concisos [TEL2004].

Devido a estas dificuldades, o XP aborda o processo de documentação de uma forma diferente.

Como já discutido na seção 2.3, a metodologia XP parte da premissa de que o cliente não possui um conjunto de requisitos previamente bem definido. Por esse motivo a metodologia do processo busca oferecer suporte para que as eventuais modificações futuras sejam incorporadas da melhor maneira possível, ou seja, com relação aos custos de alteração o XP considera uma curva distinta àquela utilizada pelos processos tradicionais [BEC2000b], como é mostrado na figura 3.2.

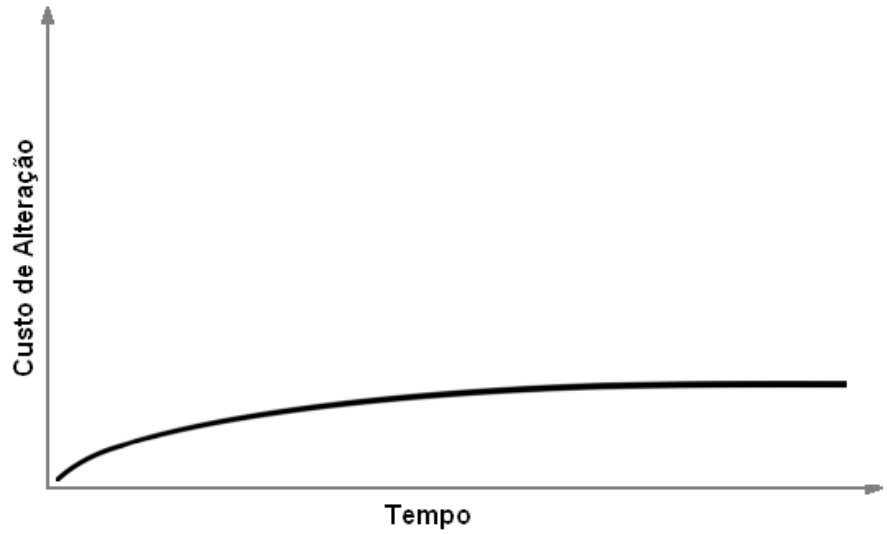

Figura 3.2 – Curva de custo de alteração de software em um projeto XP [BEC2000b].

Com relação ao processo de modelagem, este ocorre sob demanda e de modo basicamente informal, procurando produzir o mínimo possível de documentação antes de se chegar ao código. Em compensação, preferencialmente é utilizada a comunicação face-a-face como fonte de informação para o desenvolvimento das funcionalidades.

Geralmente, no processo XP a modelagem é feita de forma coletiva, através do uso de quadros brancos, onde os diagramas são construídos e discutidos por todos os membros da equipe [EXT2007]. Para persistir as informações que são expostas nos diagramas, pode se

usar uma câmera fotográfica digital, permitindo assim que o planejamento passado seja revisto posteriormente [TEL2004].

Um conjunto básico de artefatos para o XP

No XP, não temos um conjunto obrigatório de artefatos que constituem a documentação do processo. Cada projeto é único, tendo suas próprias necessidades de arquivamento de informações ao longo de seu ciclo de vida, principalmente na fase de operação e manutenção. No entanto, TELES [TEL2004] descreve um conjunto reduzido de itens que pode ser adotado como inspiração inicial para a definição da documentação do projeto em questão.

- Estórias

"*A estória é a unidade de funcionalidade em um projeto XP. O progresso é demonstrado através da entrega de código testado e integrado que implementa uma estória*." [BEC2000b]

O conjunto completo de estórias compõe a documentação dos requisitos em um projeto XP, sendo assim, BECK [BEC2000b] enumera alguns critérios que devem ser levados em consideração no momento da criação destes artefatos.

Uma estória deve ser legível tanto para os clientes quanto para os desenvolvedores, além de ser pequena o suficiente de forma a oferecer a oportunidade de implementação de um conjunto delas durante uma iteração. É aconselhável que a coleta de informações para as estórias seja feita de maneira simples, de forma a se evitar treinamentos dispendiosos em engenharia de requisitos.

As estórias devem ser escritas pelo cliente em cartões, em texto puro e simples, e no caso de ser necessário o armazenamento das estórias em computadores, estes devem favorecer a impressão das estórias nos cartões. Desenvolvedores não escrevem estórias, pois deste modo evita-se o hábito que muitos programadores possuem de tentar prever futuras necessidades, garantindo-se assim que nenhuma funcionalidade extra (desnecessária) seja produzida sem o consentimento do cliente. No entanto, os desenvolvedores possuem a liberdade de sugerir estórias.

Quanto mais curta a estória for, melhor. O ideal é que cada estória possua uma ou duas sentenças. A abstração de detalhes pode por muitas vezes deixar o desenvolvedor com incertezas, porém, a adição de muitos detalhes não garante o saneamento destas incertezas. Deste modo, o objetivo da estória é fornecer o conceito da funcionalidade e não uma especificação detalhada.

Além disso, estórias devem ser independentes, permitindo assim certa flexibilidade na questão da ordem de implementação. Elas também devem ser testáveis de alguma forma. Finalmente, cada estória deve prover algo de valor ao cliente.

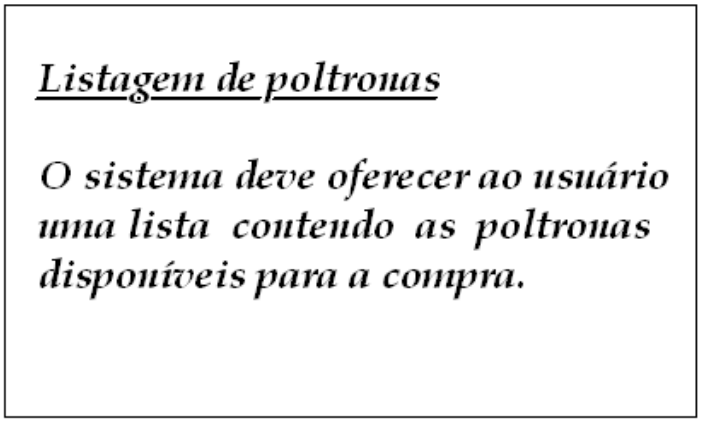

Figura 3.3 – Exemplo de estória.

- Testes de aceitação

Como citado anteriormente, cada estória deve possuir a capacidade de ser testada. Para sanar esta necessidade, os testes de aceitação fazem referência à documentação referente ao teste de cada uma das funcionalidades descritas nas estórias.

Este artefato fornece um nível de detalhamento maior que o oferecido pelas estórias. Isso por que eles devem proporcionar o máximo de informações aos desenvolvedores que irão efetuar manutenção no sistema, além de dar ao cliente a confiança de que a aplicação tem as características requeridas e que elas comportam-se corretamente.

Estes testes adotam, essencialmente, a técnica de teste da caixa-preta [CAR2004], onde os testes são planejados a partir da especificação, não levando em consideração a implementação interna das funcionalidades. Este tipo de artefato também é tido como um "contrato" entre os desenvolvedores e o cliente. Sua preservação ao longo do ciclo de vida da

aplicação juntamente com um relatório das alterações feitas mediante alteração dos requisitos garante que não houve quebra no contrato.

Os testes de aceitação são importantes por que, resumidamente, estes oferecem suporte em três questões importantes no processo de software:

- Capturam os detalhes dos requisitos do cliente e medem quão bem o sistema atende a estes requisitos;
- Expõem os problemas que os testes de unidade escondem;
- Provêem uma definição de como o sistema desenvolvido "é".

Para a aplicação dos testes de aceitação, é fundamental a especificação de um plano de testes, podendo este plano ser composto basicamente por um *checklist* contendo todos os requisitos funcionais da aplicação.

No entanto, é quase impossível para a equipe de desenvolvimento a previsão de como o cliente usará o software que está sendo desenvolvido. Informações aparentemente claras ao desenvolvedor podem ser ininteligíveis para um usuário em campo, podendo assim as instruções de uso serem mal interpretadas. Por este motivo, é importante que os testes de aceitação sejam realizados pelo cliente a fim de validar os requisitos de forma segura.

Somente após a execução e aprovação de todos os testes de aceitação pelo cliente é que o sistema desenvolvido poderá ser entregue.

Estória	**Listar poltronas disponíveis.**	
passo	**Ação**	**Resultados esperados**
1	Entrar na "Compra de passagens"	Listar as linhas disponíveis
2	Escolher a linha "Santa Maria / Porto Alegre"	Listar os horários de partida de ônibus para o destino "Porto Alegre"
3	Escolher um horário.	Listar todas as poltronas de 01 à 40.
4	Escolher a poltrona 01	Listar todas as poltronas, exceto a 01.

Figura 3.4 – Exemplo de teste de aceitação.

- Testes de unidade

Para se implementar uma dada funcionalidade, uma ou mais classes da aplicação são codificadas, contendo os atributos e métodos que manipulam um dado objeto no sistema.

Assim como cada estória, que deve possuir um ou mais testes de aceitação a fim de se avaliar sua aceitação ou não por parte do cliente, cada uma das classes que são implementadas deve ser testada, a fim de se garantir seu correto funcionamento.

Este conjunto de testes, além de certificar o correto funcionamento dos métodos das classes, também possui a utilidade de informar ao seu leitor qual o seu funcionamento esperado, atendendo assim ao ofício de documentar e validar o funcionamento da aplicação [TEL2004].

Para a aplicação destes testes, podemos utilizar ferramentas que auxiliam na implementação e aplicação dos mesmos. Uma das ferramentas mais utilizadas para este propósito, considerando-se a programação Java, é o JUnit, que pode ser obtido no site www.junit.org. Esta ferramenta também pode ser integrada aos ambientes de programação mais utilizados na atualidade, tais como o IntelliJ Idea da JetBrains e o Eclipse desenvolvido pela Eclipse Foundation. A última versão do IDE da JetBrains já possui o JUnit versão 3.8.1 integrado.

- Documentação de código

A documentação de código consiste em um artefato construído de forma automática, através da utilização de ferramentas que extraem as informações a partir das assinaturas das API's implementadas e suas respectivas definições contidas no código-fonte, em forma de comentários que utilizam uma sintaxe padronizada.

Um exemplo de ferramenta que gera este tipo de artefato é o JavaDoc, que é distribuído juntamente com o ambiente de desenvolvimento Java SDK (Standard Development Kit). Outras linguagens como Python e C++ também possuem ferramentas similares.

Este tipo de documento é de extrema importância, pois ela provê uma base de informações referentes às interfaces das classes e métodos implementados em um dado código, facilitando o entendimento e a utilização da API construída para a implementação do sistema.

- Modelo de classes

O modelo de classes ou diagrama de classes consiste em uma representação da estrutura e das relações entre as classes que servem de modelo para os objetos. Este tipo de modelo é muito útil para o sistema, pois define todas as classes que o sistema possui (ou necessita possuir) e é a base para a construção de diversos outros tipos de diagramas concordantes com o padrão UML, tais como os diagramas de comunicação, seqüência e estados [DIA2007].

Com a tecnologia atual, é possível que tais diagramas sejam criados tanto manual como automaticamente, através da aplicação de ferramentas de engenharia reversa sobre o código-fonte. No XP, recomenda-se que sempre que possível tais ferramentas de engenharia reversa de diagramas sejam utilizadas de modo a agilizar o processo de documentação, visto que a confecção de diagramas em geral consome um tempo que poderia ser desprendido em outras tarefas consideradas centrais, como a codificação e testes.

No entanto, a aquisição de tais ferramentas pode ser inviabilizada devido aos custos elevados das mesmas. No caso de o processo de engenharia reversa de código não puder ser adotado, é recomendada a adoção de uma documentação mais leve, abordando apenas os detalhes conceituais tais como as classes e seus relacionamentos. Este tipo de abordagem deve-se ao fato de que o XP se utiliza do processo de *refactoring* continuamente, fazendo com que atributos e métodos sejam muito instáveis. Seguindo a abordagem proposta pelo XP, problemas de gasto de tempo excessivo na atualização e sincronização de modelos serão evitados, oferecendo assim uma visão geral do sistema.

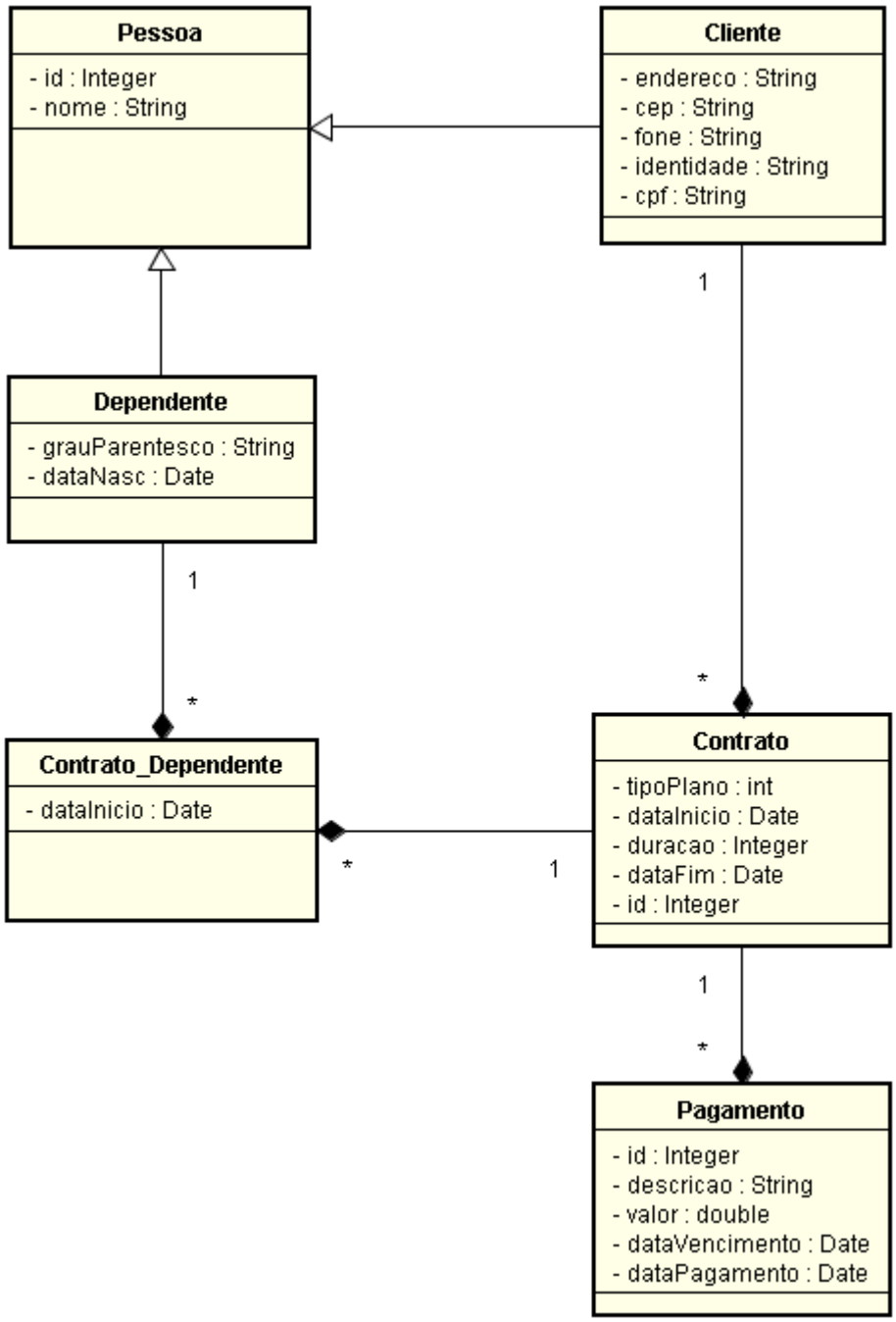

Figura 3.5 – Exemplo de modelo de classes.

- Modelo de dados

O modelo de dados é um artefato que tem por finalidade organizar os dados que serão persistidos em um banco de dados através da definição de um conjunto de conceitos para a representação desses dados.

Existem diferentes níveis de abstração dos dados que são utilizados para a construção deste tipo de documento:

- Modelos conceituais;
- Modelos lógicos;
- Modelos físicos.

Assim como ocorre com o modelo de classes, o caminho indicado aqui segue a mesma sugestão apresentada para o modelo de classes, ou seja, a equipe de desenvolvimento deve fazer as alterações necessárias no decorrer da iteração e, se possível, fazer uso de alguma ferramenta que tenha a capacidade de fazer a engenharia reversa a partir das tabelas do banco de dados.

Porém, caso esse tipo de ferramenta não esteja disponível ou, até mesmo, se o cliente exigir que um modelo de dados seja gerado antes de as alterações serem aplicadas efetivamente, recomenda-se a utilização de modelos de nível de abstração conceitual. No entanto, caso a construção de um modelo físico de dados seja uma exigência, por conta de normas internas do cliente, isso também não será problema visto que as alterações no banco são muito menos freqüentes que a alteração de código, sendo que o tempo desprendido para manter um modelo de dados consistente tende a ser pequeno.

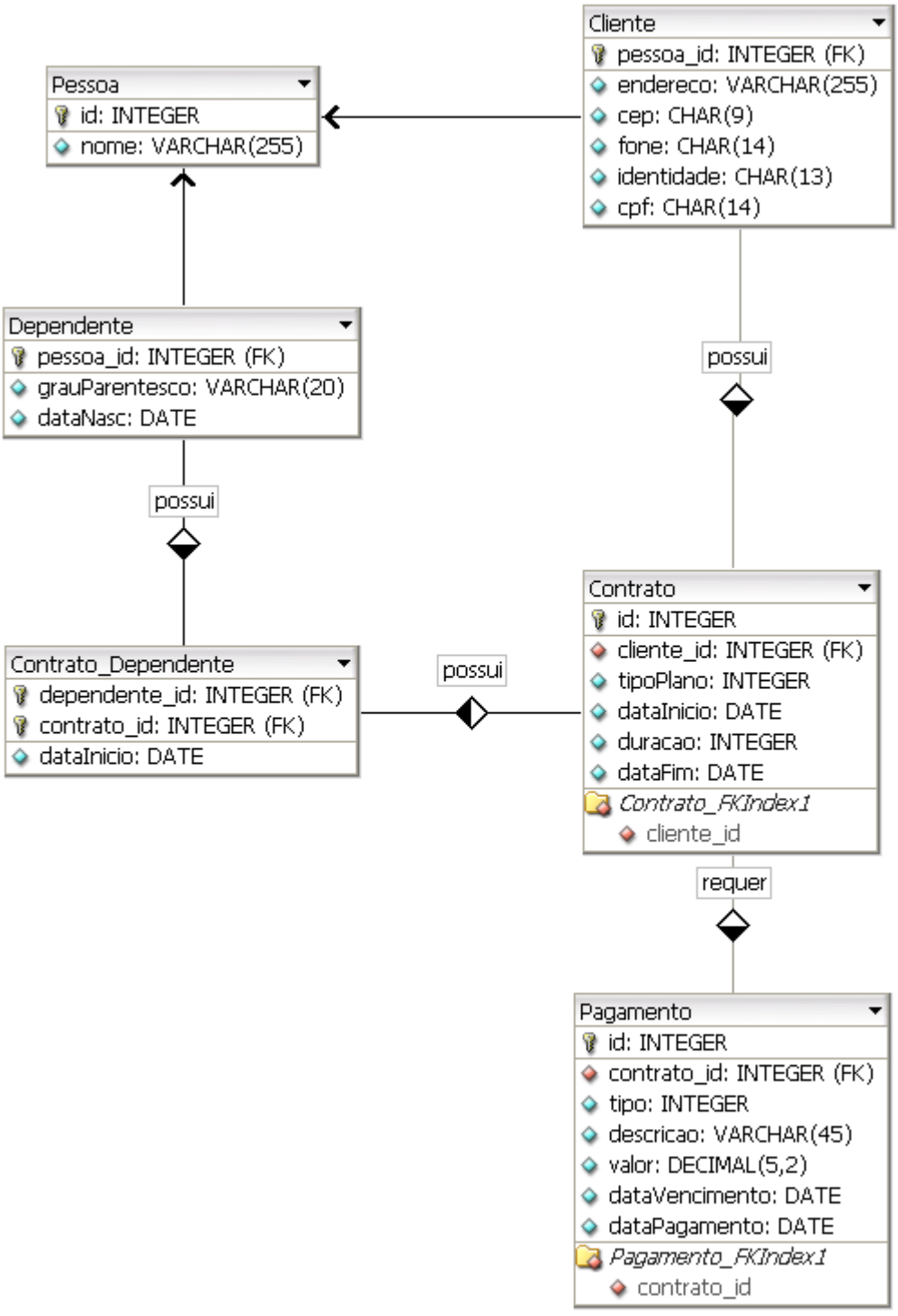

Figura 3.6 – Exemplo de modelo de dados.

- Processos de negócio

Este documento não é um documento técnico. Trata-se de um elemento que descreve os processos de negócio, em um nível conceitual, que estão incorporados no sistema desenvolvido, visando assim contextualizar os usuários ainda inexperientes que estejam manuseando o sistema.

Como tais processos evoluem muito rapidamente no decorrer do projeto, estes são muitos instáveis. Sendo assim, este artefato deve ser atualizado sempre ao término das iterações e deve conter apenas os processos que são apoiados pelo sistema. Processos que ainda não estejam incorporados ao software devem ser descritos em um documento aparte, fazendo com que este sirva como fonte de inspiração para os clientes, na criação de novas estórias.

- Manual do usuário

O manual do usuário é um item que mostra como cada um dos processos de negócio foi mapeado para dentro do sistema, indicando a seqüência de passos que um usuário deve executar para se efetuar uma determinada operação no software produzido. Esta seqüência de passos necessária em cada processo de negócio pode ser demonstrada através de imagens das telas contendo indicações de cada ação que deve ser tomada pelo usuário, a fim de se obter a resposta em cada uma delas.

- Acompanhamento diário

O acompanhamento diário trata-se de um documento gerencial que tem por finalidade auxiliar a organização e o gerenciamento das pendências do projeto. A adoção deste documento facilita o acompanhamento da evolução diária do projeto por parte do gerente, permitindo que ele exija o cumprimento das tarefas pendentes pelos respectivos responsáveis.

- Acompanhamento de projeto

O acompanhamento de projeto consiste em outro documento gerencial que objetiva concentrar informações a respeito do andamento do projeto e reportar estas informações ao cliente a cada semana. Ele relaciona as estórias que fazem parte da iteração corrente, as estórias que já foram concluídas até então, os problemas encontrados, os motivos dos atrasos, caso exista, além de outras informações que se mostrarem úteis na visão do cliente.

Este item também é importante por que ele é utilizado pelos gestores do projeto para que estes possam monitorar e reorganizar as prioridades correntes.

- Fotos

Como foi descrito anteriormente, é comum a utilização de quadros e murais para o planejamento momentâneo e para o compartilhamento de informações. No entanto, essas informações possuem uma alta volatilidade e instabilidade devido a freqüente modificação do conteúdo destes quadros expositivos.

Nestes quadros, por exemplo, é construída a modelagem, que é desenvolvida por toda a equipe em conjunto. O uso destes quadros é muito importante, pois este tipo de mídia permite um compartilhamento eficaz das informações prioritárias de certos aspectos do desenvolvimento em um determinado processo do desenvolvimento. No entanto, embora os quadros e murais sejam um instrumento eficaz no apoio às decisões da equipe, eles possuem um problema que é a falta de persistência das informações neles contidas.

Uma maneira simples e eficaz de se persistir estas informações nestes tipos de mídia é o arquivamento de fotos do conteúdo diário destes quadros e painéis através da utilização de uma câmera fotográfica digital e armazena-las em um servidor de documentação, onde podem ser arquivados todos os artefatos referentes à documentação da modelagem e implementação do software em desenvolvimento.

A concentração da documentação do projeto em um servidor não somente facilita a questão do arquivamento dos artefatos como também pode tornar o acesso a tais itens pelos seus desenvolvedores mais eficiente.

3.3 Agile Modeling

Como vimos até agora, a metodologia XP também proporciona uma grande flexibilidade no que se refere aos artefatos que compõem o conjunto documentativo dos sistemas desenvolvidos, porém, este não é o único esforço empregado neste contexto.

No intuito de combinar os padrões da indústria e as práticas de modelagem mais utilizadas com o pensamento Ágil, Scott W. Ambler, atual líder de práticas de desenvolvimento ágil dentro da IBM concebeu uma metodologia de modelagem denominada *Agile Modeling* (AM).

O *Agile Modeling* representa uma nova metodologia utilizada para supervisionar processos de software, no entanto, assim como o Extreme Programming, esta nova metodologia se baseia em um conjunto de valores, princípios e práticas [AMB2002, BAI2002, MEN2005], que quando aplicadas de forma concisa oferecem ao processo o desenvolvimento eficaz de modelos e documentos de um sistema.

Algumas das suas características mais interessantes que podem ser citadas são:

- Mesmo consistindo em uma nova metodologia por si própria, o AM pode ser empregado em conjunto com outra metodologia-base tal como XP ou RUP.
- Permite que cada usuário adapte a metodologia de modo a atender suas necessidades locais.
- Seu conjunto básico de valores, princípios e práticas em geral estão fortemente relacionados às do XP.
- Promove a criação de uma quantidade “apenas o suficiente” de artefatos de modelagem e documentação do sistema.

O processo de modelagem é um importante instrumento de suporte à comunicação entre os membros de uma equipe. No XP, cada iteração começa com a definição de uma coleção de estórias e tarefas. No entanto, estórias são artefatos que documentam as necessidades de um cliente em um alto nível. Com a evolução da iteração, uma quantidade cada vez maior de informações toma espaço do quadro de modelagem, provendo esclarecimentos mais amplos às omissões destes instrumentos de alto-nível.

Os desenvolvedores então utilizam os mais variados tipos de diagramas ou outras notações de modo a expressar as idéias de mais baixo nível. No AM, todos estes artefatos, sejam eles expressos via papel ou quadros, são freqüentemente descartados assim conseguem

completar sua função. Entretanto, no caso de estes artefatos ainda possuírem a capacidade de prover alguma utilidade no decorrer do processo, eles podem ser mantidos.

A seguir, na tabela 3.3, apresenta-se o conjunto de valores na qual o AM se baseia e sua respectiva contribuição para com a metodologia.

Tabela 3.3. Os valores do *Agile Modeling* [BAI2002].

Valor	**Descrição**
Coragem	No decorrer do projeto, inúmeras decisões deverão ser tomadas. O usuário dever ser capaz de avaliar e reformular ações, além de ter coragem em admitir que sua modelagem possa estar errada ou insuficiente e aceitar novas direções.
Comunicação	A modelagem existe para dar suporte à comunicação entre os membros da equipe. Se certo artefato não está conseguindo desempenhar esta função, troque-o por outro que o faça.
Feedback	Diagramas facilitam o entendimento do cliente, facilitando o *feedback*. Os modelos devem sempre ser validados e explanados aos clientes.
Humildade	Os modeladores devem ter em mente que eles não sabem tudo, assim como cada pessoa possui sua própria especialização. Eles devem ser capazes de aceitar as críticas e incorporar sugestões.
Simplicidade	Manter os modelos o mais simples possível, e caso estes não reflitam a realidade ou possuam falhas estes devem ser alterados sem que muita complexidade seja adicionada.

Como podemos observar, fica evidente que o AM teve seus valores inspirados nos valores na metodologia XP. Como veremos na tabela 3.5, esta correlação não está presente apenas nos valores, mas também nas práticas adotadas pelo AM. No entanto, o AM, além de ser guiado por valores e práticas assim como o XP, também fornece um conjunto de princípios que também ajudam a dar suporte às práticas da metodologia. Apresentamos a seguir, na tabela 3.4 uma listagem dos princípios fundamentais, e após, na tabela 3.5, a listagem das práticas executadas no AM, ambas com uma breve explanação a respeito de cada um de seus itens.

Tabela 3.4. Os Princípios Fundamentais do *Agile Modeling* [BAI2002].

Princípio	**Descrição**
Assumir a simplicidade	Manter os modelos o mais simples possível. Modelar hoje apenas o necessário e acreditar que é possível remodelar amanhã se necessário.
Suportar mudanças	Mudanças ocorrerão assim que os requisitos forem entendidos mais claramente. Deve se aceitar as mudanças e ter a coragem de refazer o que é necessário.
Viabilizar esforços futuros como objetivos secundários	Outros podem precisar estender o projeto após seu término. Por isso, produza apenas a documentação necessária para que a próxima equipe possa estender o projeto com facilidade e segurança.
Mudanças incrementais	Os modelos não precisam estar perfeitos logo no início. Devem-se fazer pequenas correções ao longo do tempo quando requeridas.
Maximizar investimentos dos *Stakeholders*	A equipe produz software para maximizar retorno para o cliente. A modelagem e documentação devem ser criadas no intuito de adicionar este valor ao software de modo que os investimentos sejam mais freqüentes e maiores.
Modelar com propósito	Os modelos e documentos devem ser criados com um objetivo a ser seguido. Deve-se saber para quê os artefatos estão direcionados e o que eles querem expor.
Utilizar múltiplos modelos	Existem várias maneiras de se expor uma solução. Escolha aquela que caiba na situação em questão. Por exemplo, modelos de dados são direcionados à equipe de banco de dados.
Trabalho com qualidade	O projeto deve garantir a satisfação dos clientes frente às suas expectativas.
Feedback rápido	Quanto mais rápida for a reação frente a um aspecto da modelagem, mais rápida será a adaptação da mesma.
Software é o objetivo principal	Modelos e documentos não são nada além de artefatos de auxílio na busca pelo objetivo final. A documentação e modelagem devem facilitar a evolução do software.
Travel light	*Traveling light* significa que você possui apenas a quantidade suficiente de documentação para o processo. Sua falta fará com que a equipe se perca; seu excesso resultará no esquecimento do objetivo principal – escrever software.

Tabela 3.5. Práticas Fundamentais do *Agile Modeling* [BAI2002].

Prática	Descrição
Participação ativa dos *Stakeholders*	O AM provê a participação aos utilizadores que possuem a capacidade de dar informações relevantes ao processo além de tomar decisões quanto a alocação dos recursos disponibilizados.
Aplicação dos artefatos corretos	Existem muitos tipos de artefatos que podem ser utilizados para exprimir uma idéia. Deste modo, deve se procurar utilizar o artefato correto, de acordo com a situação encontrada.
Posse coletiva	A equipe por completo tem a posse dos artefatos. Esta prática é executada com a utilização de instrumentos que facilitem o compartilhamento das informações como quadros e servidores públicos de documentação.
Considerar testes	Na medida em que os artefatos são criados, deve ser analisada a possibilidade de se efetuar testes nos modelos, e como estes testes serão feitos.
Criar diferentes modelos em paralelo	Na busca pelo entendimento de um problema, deve ser considerada a possibilidade de se utilizar mais de um tipo de notação.
Criar conteúdo simples	Os modelos devem conter apenas informações suficientes para que o modelo possa cumprir seu propósito de facilitar o entendimento do problema.
Descrever modelos de forma simples	Quando se estiver modelando, não se deve utilizar operações de sentido complexo ou obscuro. O ideal é utilizar um subconjunto de operações da notação-base tomada.
Mostrar os modelos publicamente	Mostrar os modelos publicamente enfatiza a natureza de coletividade e dá ao cliente um definitivo senso de direção.
Iterar para outro artefato	Quando o modelo em utilização não estiver conseguindo exprimir as informações com clareza, deve-se iterar para outro tipo de artefato que o faça apropriadamente. Muitas vezes tudo o que é preciso, é apenas uma adaptação da notação para que a comunicação seja facilitada.
Modelar de forma incremental	Não há a necessidade de se prever todo um cenário. A modelagem incremental se ajusta ao contexto de desenvolvimento iterativo: modelar, comunicar, refinar, e remodelar.
Modelar em equipe	O XP e o d AM são esforços em equipe onde a sinergia do grupo amplifica a eficiência da equipe.
Provar com código	Às vezes, a melhor maneira de se validar certa modelagem é através da escrita do respectivo código, afinal, este é o objetivo primário da equipe.
Utilizar as ferramentas mais simples	Desenvolvedores em geral tendem a modelar em papéis ou quadros e depois transcrever seus modelos para uma ferramenta formal de design. Modeladores Ágeis não têm medo apenas digitalizar seus modelos via utilização de scanners e câmeras digitais. Modeladores Ágeis usarão ferramentas CASE mais complicadas apenas quando isto se mostrar necessário.

O AM surgiu com dois objetivos: proporcionar a disciplina no processo de modelagem e documentação de metodologias ágeis como o XP, assim como proporcionar a agilização de processos mais rígidos como o RUP, por exemplo. A similaridade entre os valores e práticas do AM compete à metodologia uma facilidade para associá-la aos projetos XP.

A seguir, na tabela 3.6, temos a apresentação das relações existentes entre o AM e o XP.

Tabela 3.6. Relação entre as práticas AM e XP [BAI2002, MEN2005].

Prática AM	**Prática XP relacionada**
Participação ativa dos *Stakeholders*	Correspondente à prática de "Cliente presente".
Aplicação dos artefatos corretos	Não há correlação direta.
Posse coletiva	Correspondente à prática de "Código coletivo".
Considerar Testes	Correspondente à prática de "testes".
Criar diferentes modelos em paralelo	Não há correlação direta.
Criar conteúdo simples	Complementar à prática de "*Design* Simples".
Descrever modelos de forma simples	Também complementar à prática de "*Design* Simples".
Mostrar os modelos publicamente	Complementar à prática de "Código coletivo" e ao valor "comunicação".
Iterar para outro artefato	Não há correlação direta.
Modelar de forma incremental	Correspondente à característica de desenvolvimento incremental.
Modelar em equipe	Correspondente à prática de "programação em pares".
Provar com código	Correspondente à ênfase à codificação.
Utilizar as ferramentas mais simples	Complementar ao valor "Simplicidade".

A quantidade de princípios e práticas existentes no processo AM é muito extensa, porém uma das características mais interessantes desta metodologia está no fato de que o usuário pode construir um subconjunto personalizado, apropriado à individualidade do seu projeto.

Como se pode observar, todo este conjunto de informações apresentados nesse capítulo demonstram que há esforços no que diz respeito à criação de uma documentação de qualidade no XP. Também fica evidente que o XP, além de proporcionar flexibilidade na produção do software, também o faz na parte de modelagem e documentação, inspirando a criação de outras metodologias que podem ser utilizadas de forma cooperativa.

3.4 Discussão

Como pôde ser observada na seção 3.1, a documentação é fundamental para a fase de manutenção dentro do ciclo de vida de um software. Além disso, os dados estatísticos apresentados nos estudos de SOUSA *et. al* [SOU2004a, SOU2004b] (tabelas 3.1 e 3.2) mostram que os artefatos mais importantes em um projeto de software são o código-fonte,

evidentemente; os comentários nele introduzidos (reforçando a importância da padronização de código); o modelo de dados lógico e físico; e os diagramas de classes e casos de uso.

Destes artefatos apresentados na pesquisa, citados anteriormente, que foram indicados pelos profissionais participantes como sendo os mais importantes (itens do topo da tabela 3.1), apenas o diagrama de casos de uso não foi abordado no conjunto que compôs a proposta inicial feita por TELES [TEL2004], demonstrando assim uma fundamentação válida para este conjunto básico de artefatos.

Em relação aos dados apresentados na tabela 3.2, podemos observar o risco que a falta de documentação pode representar para a manutenção de software. Dos 24 artefatos postos a prova, 14 foram utilizados pelos profissionais participantes da pesquisa em questão. Também podemos observar que, em geral, os artefatos ditos mais importantes na primeira pesquisa (Tabela 3.1) foram os que mais se mostraram disponíveis nos softwares mantidos o longo da vida profissional dos desenvolvedores. Ainda, observa-se que a maioria dos artefatos considerados não importantes na primeira pesquisa não são efetivamente utilizados pelos mantenedores.

Uma exceção ocorreu em relação aos planos de testes (unitários e de sistema), os quais na primeira pesquisa não obtiveram um alto grau indicações como sendo muito importantes, mas que na segunda pesquisa apresentou um alto índice de disponibilidade. Outra exceção ocorrida foi com relação ao artefato “dicionário de dados”, que não foi muito apreciado pelos profissionais na primeira pesquisa, porém na segunda demonstrou um alto índice de utilização e disponibilidade.

Além disso, como se pôde ver na seção 3.2, o AM reforça o pensamento que a modelagem e documentação não apenas são importantes para a fase de manutenção, após a conclusão do software, como também para aumentar a qualidade e a eficiência da comunicação entre os desenvolvedores e os *stakeholders*. Esta metodologia auxiliar, que foi concebida com o intuito de combinar as práticas de modelagem de software mais utilizadas com o pensamento Ágil, propõe a aplicação de um conjunto de valores, princípios e práticas muito semelhantes àquelas que são empregadas na metodologia XP, fazendo com que haja um comportamento disciplinado em relação à modelagem e documentação do processo XP sem que esta perca a sua leveza e flexibilidade.

Assim como foi apresentado na seção 3.1, o *Agile Modeling* aborda a documentação de uma maneira muito flexível, onde os artefatos a serem utilizados para arquivar as informações mais importantes, pertinentes ao software em questão, são escolhidos pela equipe juntamente com o cliente. No entanto, o conjunto de artefatos apresentado por TELES

[TEL2004] se mostra bastante eficaz, tendo inclusive a sua utilidade justificada pelos dados apresentados pelo estudo de SOUSA *et. al* [SOU2004a, SOU2004b].

Através da ligação dos dados apresentados até então, podemos propor uma abordagem documentativa que se mostraria válida para a maioria dos projetos desenvolvidos sob supervisão da metodologia XP: a adoção das idéias (valores, princípios e práticas) do AM, aliado ao conjunto documentativo inicial proposto na seção 3.2, acrescido dos artefatos de testes de unidade e de sistema e o diagrama de casos de uso. Deste modo, tem-se um processo documentativo e de modelagem com base em ações disciplinadas no XP, além de se formalizar um conjunto essencial de artefatos capaz de enquadrar os aspectos documentativos do processo de software que possuem a capacidade real de facilitar a comunicação no grupo e que também terá uma utilização efetiva na fase de manutenção do software.

4 EXPERIÊNCIAS

Em sua grande maioria, os valores e práticas tanto do XP assim como do AM são baseados em princípios coletivos, ou seja, possuem aspectos que exigem a participação de uma equipe real, tornando limitada a sua abordagem de maneira individual.

Este capítulo visa contemplar, dentro do possível, alguns aspectos sobre as experiências vivenciadas a respeito do processo documentativo e de modelagem do XP. Para isso, a aplicação que será apresentada aqui visa atender as necessidades de uma típica Clínica Odontológica.

Este sistema destina-se a efetuar o controle dos aspectos mais comuns envolvidos na administração de um consultório odontológico, tais como controle de mensalidades dos planos odontológicos, cadastros de clientes e seus respectivos dependentes, entre outros eventuais aspectos que poderão se mostrar necessários à cliente no decorrer do desenvolvimento.

4.1 A fase de pré-início de um projeto XP

Primeiramente, uma das coisas mais importantes antes de se começar a abordagem do desenvolvimento é o planejamento e a organização do ambiente de trabalho. A correta organização é elemento chave para que a primeira iteração, considerada por muitos a mais importante, evolua com tranqüilidade.

Fazer uma organização do equipamento a ser utilizado, assim como a instalação e configuração das ferramentas com antecedência em relação ao começo das atividades do projeto é uma atividade muito recomendada.

Deste modo, foi organizado (instalado e devidamente configurado) o conjunto de ferramentas que viria a ser utilizado no decorrer do projeto.

- Sistema Operacional

Para o desenvolvimento da aplicação, foi levado em conta o sistema operacional Windows XP da Microsoft, que corresponde hoje o sistema operacional de maior utilização ao redor do mundo, porém, com a atual popularização dos sistemas baseados em software livre, o incentivo de empresas tradicionais no mundo da informação a estes sistemas, e até mesmo a crescente migração de entidades tanto públicas como privadas para sistemas Linux, também será valorizado o suporte à compatibilidade da aplicação implementada a estes sistemas operacionais.

- A linguagem de programação

Java é uma linguagem de programação orientada a objeto que foi projetada tendo em vista a portabilidade. Destacam-se também outras vantagens apresentadas pela linguagem, tais como a sintaxe similar a Linguagem C/C++, facilidades de Internacionalização, a distribuição com um vasto conjunto de bibliotecas, facilidades para criação de programas distribuídos e multitarefa e a desalocação de memória automática por processo de coletor de lixo (*garbage collector*);

- O Ambiente de Desenvolvimento Integrado

O Ambiente de Desenvolvimento Integrado, também referenciado como IDE, é uma das ferramentas mais importantes na condução do desenvolvimento de um projeto de software, pois ela deve fornecer uma diversidade de facilidades ao programador no que se refere à assistência de codificação e de depuração do código produzido. Na lista de características comuns obrigatórias em um bom IDE na atualidade, podemos citar compleção de código, inspeção de código e assistente de importação. Para o desenvolvimento em XP, um IDE também deve ser capaz de proporcionar facilidades para o *refactoring*.

Através dessa análise, o IDE eleito para o desenvolvimento do projeto foi o ambiente produzido pela empresa JetBrains, chamado IntelliJ Idea, na qual atualmente se encontra na versão 6.0.2. Entre as diversas funcionalidades desta ferramenta, podemos citar:

- Sistema avançado de *Code Completion*;
- Sistema de Inspeção de código inteligente;

- Assistente de importação de pacotes otimizado;
- Ferramenta de desenvolvimento de GUIs integrado;
- Integração com a ferramenta Ant para automação de builds;
- Refactoring de código automático;
- Suporte à JavaDoc;
- Avançado suporte para e debug do código;
- Suporte a plug-ins de terceiros;
- Versões para Windows e Linux;

Esta ferramenta está disponível no site www.jetbrains.com e pode ser licenciada em diversas modalidades, incluindo-se licenças de avaliação, acadêmicas, educacionais e para projetos de código aberto (*Open Source*), além das tradicionais licenças comerciais.

- A ferramenta de testes

Como descrito anteriormente, o XP promove o teste do código produzido continuamente no decorrer do projeto. Para a execução desta importante tarefa, é utilizada a ferramenta JUnit, que pode ser obtida no site www.junit.org e integrada ao ambiente IntelliJ Idea. A última versão do IDE da JetBrains já possui o JUnit versão 3.8.1 integrado.

- As ferramentas de modelagem

Para desenvolver os diversos tipos de modelagem do sistema, será utilizado uma gama diversificada de aplicativos, nas quais podemos citar o JUDE (Java and UML Developers' Environment) da ChangeVision, DBdesigner da fabForce.net, além de plugins de modelagem embarcados na própria IDE.

- O gerador de documentação

Para a geração da documentação do código escrito, será utilizada a ferramenta Javadoc. Esta ferramenta é capaz de gerar uma documentação das APIs no formato HTML a partir dos comentários inseridos diretamente no código fonte. Esta ferramenta faz parte do pacote padrão de desenvolvimento Java.

- O Sistema de Banco de Dados

O Sistema Gerenciador de Banco de Dados a ser utilizado é o MySQL Server na sua versão 5.0. Este sistema de banco de dados se popularizou devido a sua fácil integração com o PHP, sendo atualmente um dos bancos de dados mais populares no mundo. [MYS2007]

Entre suas características principais, podemos citar:

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Java, C/C++, Python, Perl, PHP e Ruby);
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de hardware;
- Facilidade de uso;
- É um Software Livre;
- Suporte a vários tipos de tabelas (como MyISAM e InnoDB), cada um específico para um fim;

4.2 Começando o desenvolvimento

Após a conclusão dos primeiros cuidados de instalação e configuração do ambiente, iniciou-se a primeira iteração.

Deu-se início então à comunicação com o cliente e as primeiras estórias (documentos que descrevem cada uma das funcionalidades da aplicação) foram escritas. Através desse conjunto de cartões de estórias, somado ao conhecimento adquirido através do diálogo direto com o cliente, um mapa conceitual do sistema foi sendo desenhado, de forma a tornar os diálogos mais facilmente entendidos por ambos os lados, assim como para fornecer uma visão geral do sistema a ser implementado. Essa visão pode ser vista na figura 4.1 a seguir.

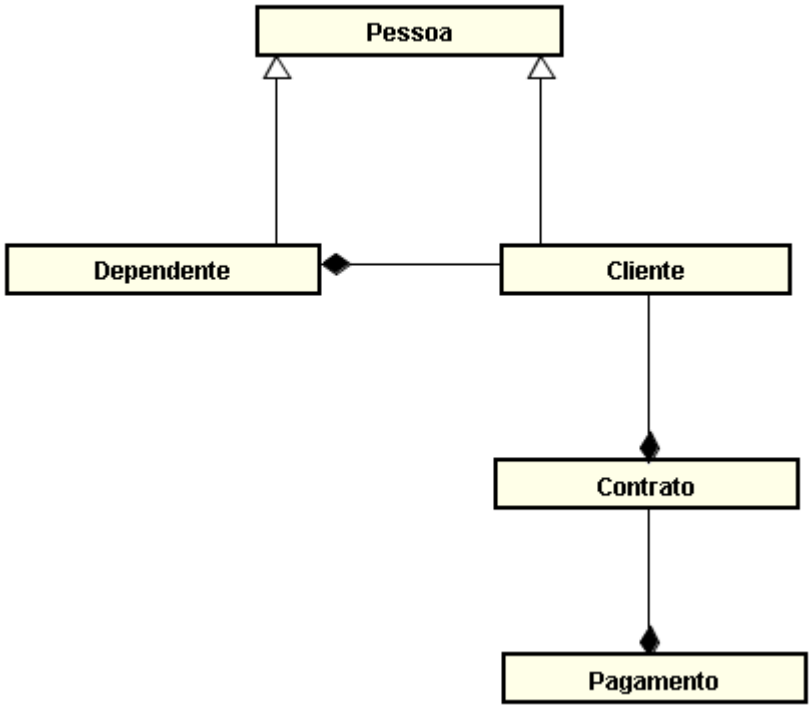

Figura 4.1 – Uma visão inicial do sistema a ser desenvolvido.

A partir da obtenção dessas primeiras estórias e dessa primeira aplicação da prática de design simples, começou-se o desenvolvimento das APIs da aplicação, formadas pelas estruturas básicas (Classes) e seus respectivos métodos, de modo a atender este conjunto inicial de requisitos. Nesta parte do projeto, também foram definidas as prioridades sobre as funcionalidades a serem implementadas.

Primeiramente, tendo como base o fato de o cliente reivindicar o software para que o mesmo faça o controle de seus clientes assim como seus dependentes, começou-se a definição das APIs e a respectiva escrita de classes de teste para as mesmas. Após isso, a implementação do código começou a tomar lugar no processo de desenvolvimento.Neste contexto, podemos citar alguns exemplos de estórias, tais como: "*o sistema deve persistir os dados do cliente no banco de dados*", "*o sistema deve sugerir o próximo código de cliente a ser utilizado*" ou ainda " *aplicação deve calcular a data de vencimento de um plano odontológico a partir da data de início e tempo de duração fornecidos pelo usuário*", entre outros.

Após a escrita de um conjunto de inicial de funcionalidades, assim como seus respectivos testes de unidade, tornou-se necessário a implementação de uma interface gráfica, para que esta facilite o entendimento de como o sistema se apresentará para o usuário, além de proporcionar um *feedback* mais substancial por parte do cliente.

Alguns modelos de interface foram então esboçados em conjunto com o cliente, como podemos observar na figura 4.2.

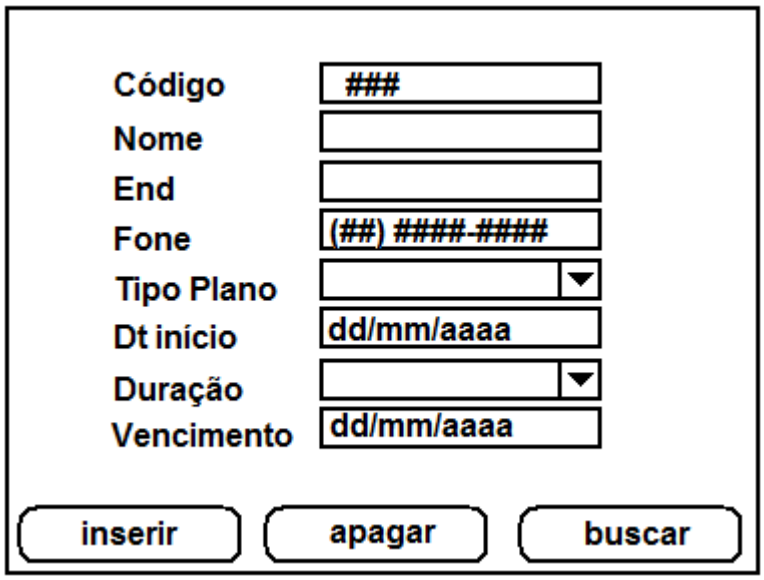

Figura 4.2 – Esboço da interface de usuário.

Estes modelos, além de servir como demonstrações de interfaces gráficas, também têm outras funções muito importantes, como por exemplo, ajudar os desenvolvedores a visualizar quais são os dados que devem ser entrados pelo usuário e de que forma eles devem ser tratados. Além disso, estas telas esquematizadas servem de inspiração na criação de novas estórias e ajudam o cliente a definir os requisitos do sistema.

Um bom exemplo da utilidade desses esquemas pôde ser observado em um momento da implementação. Após a demonstração gráfica do que viria a ser futuramente a tela de cadastro de clientes combinado com a comunicação direta com o cliente resultou em uma alteração em uma classe já escrita e em funcionamento: o cliente se deu conta que um importante campo (dado do cliente) não havia sido abordado. Prontamente este novo campo foi adicionado à classe "*Clientes*", fazendo com que diversos métodos desta classe fossem adequados à nova realidade por meio do *refactoring*.

Também através desse conjunto de entidades e seus respectivos atributos, foi possível imaginar um modelo de dados inicial para que seja possível a persistência dos dados. A seguir, na figura 4.3, temos um modelo de dados para a aplicação, contemplando as informações de cliente abordadas até o momento.

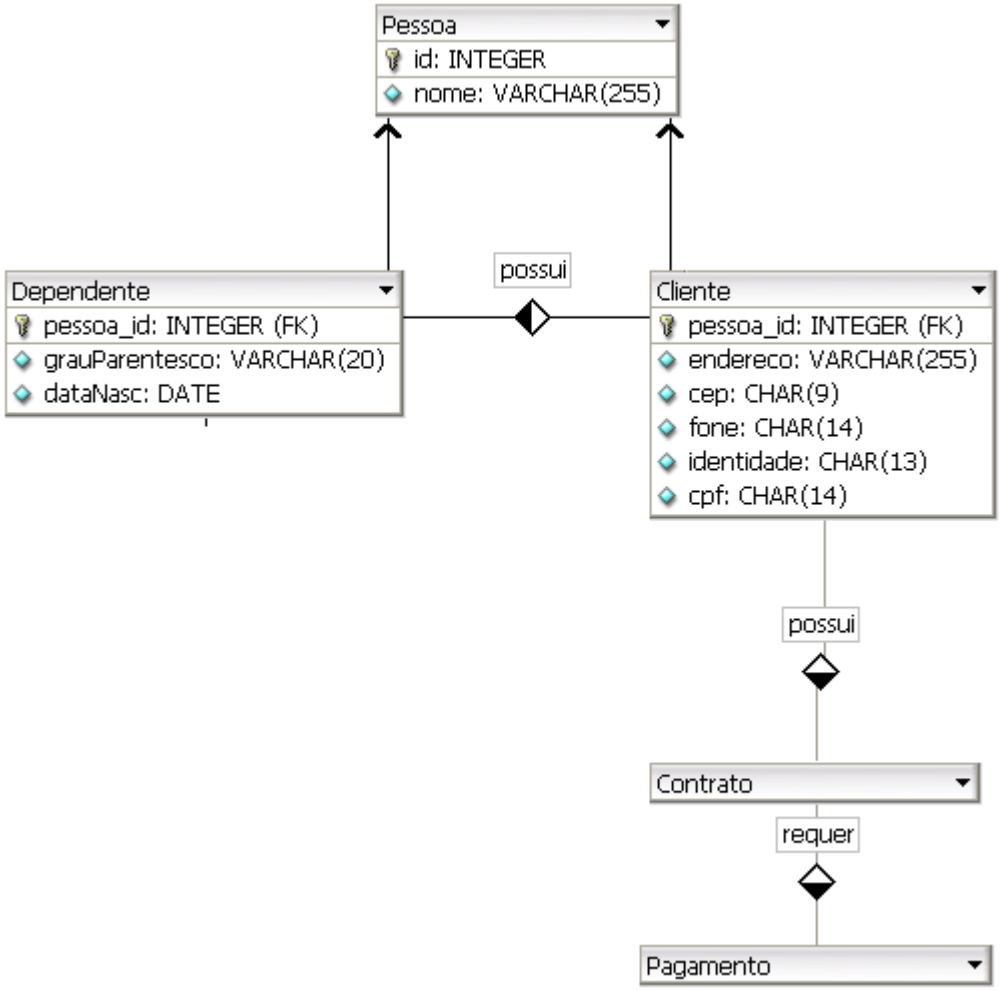

Figura 4.3 – O modelo de dados inicial.

Após alguns dias de trabalho, como resultado, possuímos um protótipo na qual o cliente já pode observar seu funcionamento e fornecer um *feedback* ainda mais valioso. A seguir, na figura 4.4, observamos a primeira tela gerada pela implementação.

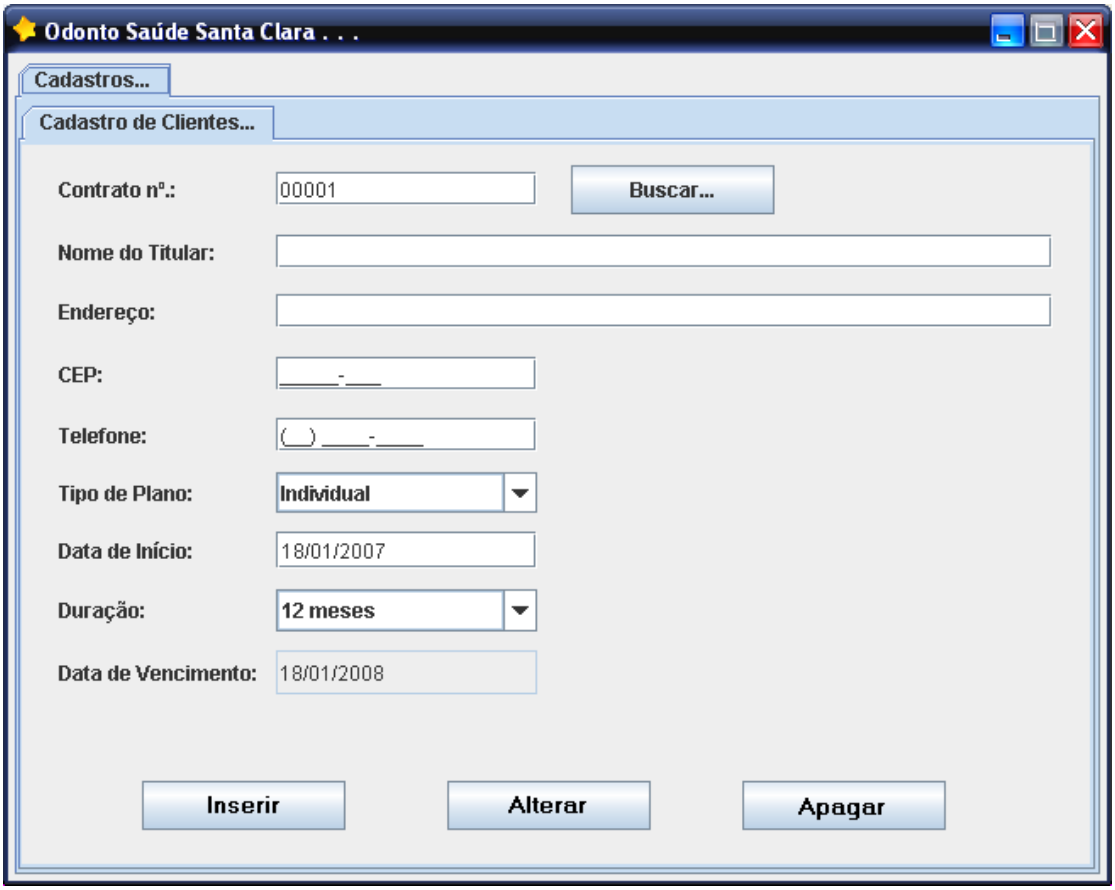

Figura 4.4 – Tela do primeiro protótipo do cadastro de clientes.

A partir deste protótipo, foram então criadas outras estórias com o intuito de melhorar a interação entre usuário e aplicação, como por exemplo, funções de validação dos dados inseridos bem como elementos visuais que indicassem tais campos inválidos.

No decorrer da iteração, os demais cartões-estória estipulados para a mesma foram implementados, dando origem ao cadastro de clientes e dependentes. A implementação e integração dessas funcionalidades trouxeram a tona outras questões até então não definidas pelo cliente, como por exemplo, de que forma a renovação de contratos deveria ocorrer?

O cliente então pensou que poderia ser uma boa característica para o sistema não somente controlar os clientes com contratos ativos como também manter um histórico de todos os contratos, clientes, dependentes e pagamentos que já passaram pelo sistema. Esta nova característica fez com que o sistema passasse por outra análise e as funcionalidades já desenvolvidas tiveram de sofrer mais um grande processo de *refactoring*. A fim de se obter a capacidade de se manter contratos inativos e ativos funcionando de forma consistente, algumas questões da relação entre clientes e dependentes tiveram de ser remodeladas, de modo que um cliente que já possuiu um contrato familiar (possuindo dependentes ligados a si)

pudesse obter um novo contrato, agora do tipo individual, sem que haja a necessidade de se remover os dependentes a ele ligados (o plano do tipo individual não aceita dependentes).

A solução para o problema descrito acima foi simplesmente utilizar o próprio contrato para fazer a relacionar um dependente a um cliente titular, ou seja, os dependentes não são mais ligados a um cliente diretamente, mas sim via contrato.

A modelagem do sistema então foi adequada para tratar dessa modificação de requisitos, se mostrando agora conforme a figura 4.5.

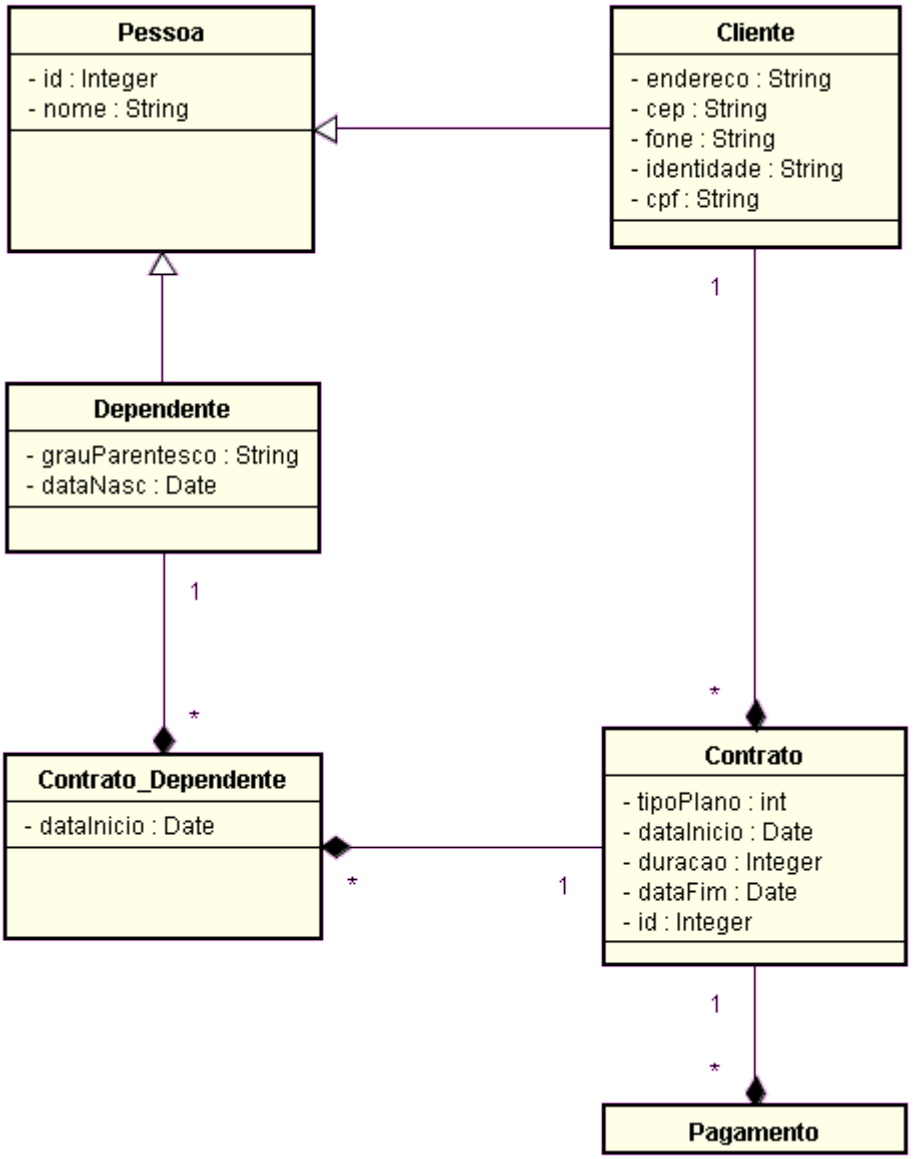

Figura 4.5 – O Diagrama de Classes após a alteração.

A modelagem de dados também sofreu alterações semelhantes, apresentando-se agora como é mostrado na figura 4.6.

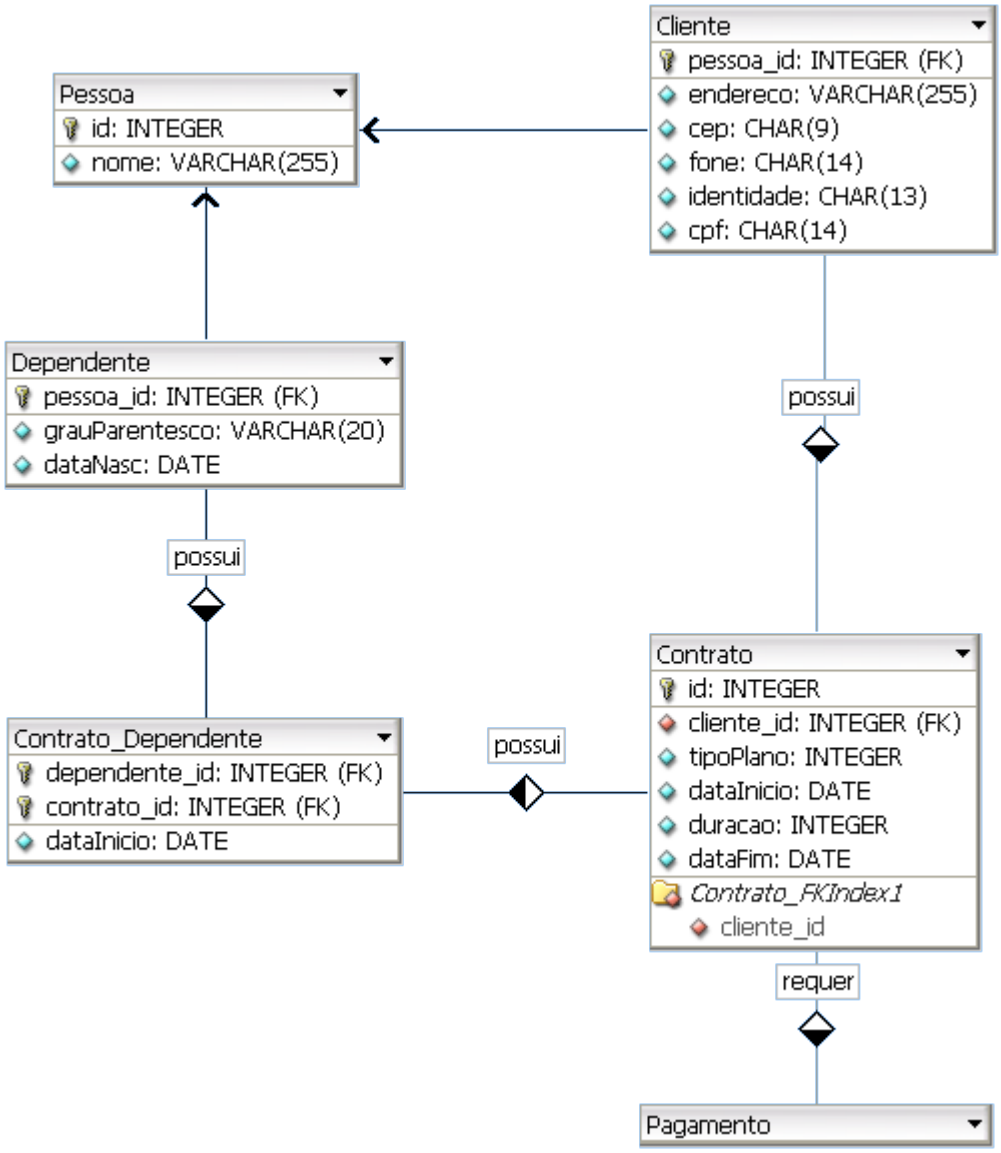

Figura 4.6 – O Diagrama de Dados após a alteração.

4.3 Resultados da primeira iteração

Ao final desta iteração, foi obtido um sistema cadastral de clientes e dependentes, assim como a definição de como um contrato se comporta.

Com o objetivo de se minimizar o tempo gasto na composição de artefatos assim como tendo em vista a limitação imposta pela implementação individual, apenas os seguintes artefatos de documentação, como foi visto até o momento, foram criados:

- Estórias;
- Modelos de dados;
- Diagramas de classes;
- Javadocs;
- Testes de unidade;
- Imagens-esboço da interface do sistema;
- Código-fonte.

A seguir, na figura 4.7, pode-se visualizar o protótipo resultante, após o término da primeira iteração do projeto.

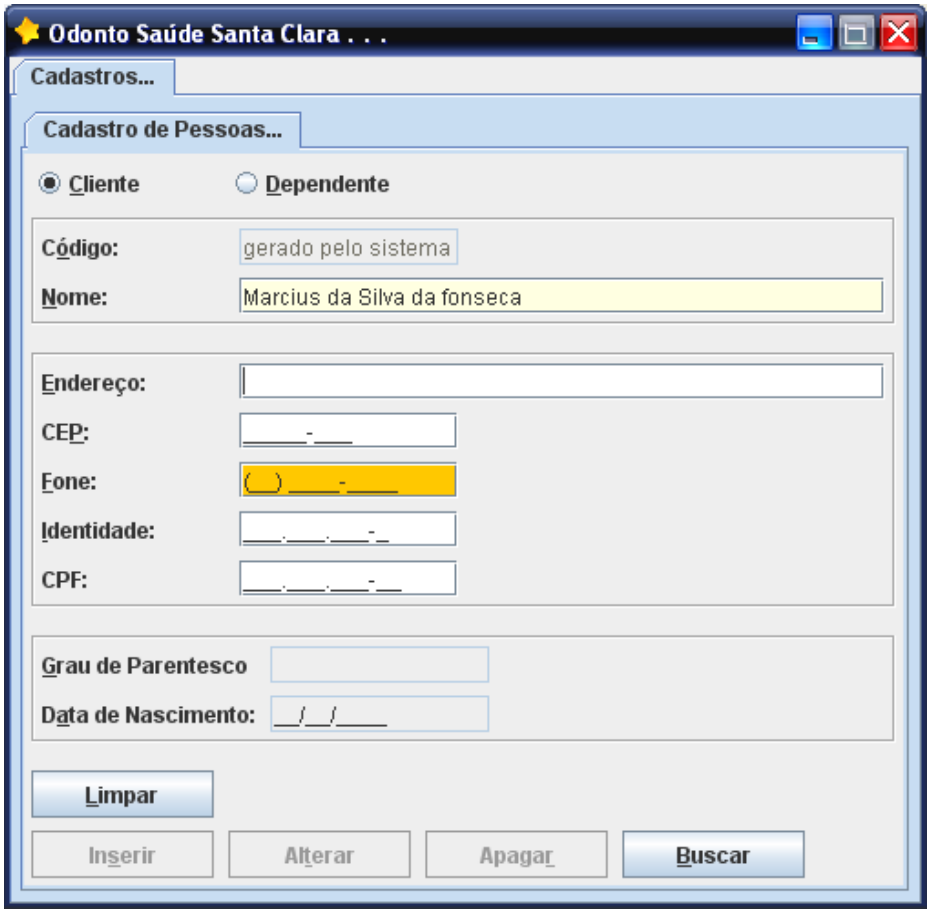

Figura 4.7 – O protótipo da aplicação após o término da primeira iteração.

5 CONCLUSÕES

As metodologias tradicionais de desenvolvimento de software partem da premissa de que os requisitos de um projeto de software podem ser coletados e descritos completamente em uma fase inicial do processo. Este tipo de abordagem geralmente se mostra como sendo equivocada, visto que no decorrer do processo, as incertezas são inerentes e as mudanças muitas vezes inevitáveis.

Descrita freqüentemente como sendo uma metodologia leve que enfatiza codificação, simplicidade e comunicação como princípios fundamentais, nos últimos anos muito tem se falado a respeito do XP. Estas discussões geralmente têm como elemento central outra característica que compete ao XP um alvo para a polêmica: a baixa ênfase imposta na documentação do processo.

Tendo estas considerações em mente, este trabalho representa uma busca de informações a cerca da metodologia de software Extreme Programming, mais especificamente sobre os métodos de documentação envolvidos na técnica.

Enquanto algumas pessoas afirmam que o formalismo rígido e a abstração são fatores importantes para a evolução de um processo de software, outras enfatizam a necessidade de se criar e utilizar métodos mais leves e flexíveis, alegando que a alta preocupação com formalismos resulta em inflexibilidade e lentidão, somando ao projeto de software certa chance ao fracasso.

No entanto, um ponto fundamental que precisa ficar esclarecido é que o XP não abandona a necessidade de se documentar a evolução e a estrutura do software produzido em seu processo. Como foi visto neste trabalho, o que ocorre é uma abordagem distinta em relação ao processo de modelagem e formalismos. O que realmente ocorre ao XP é a falta de trabalhos que evidenciem a importância da documentação nesta metodologia, assim como estudos mais experimentais que possam definir um conjunto de artefatos de documentação padrão para a metodologia.

Em um dos itens bibliográficos consultados, TELES [TEL2004] descreve um conjunto de artefatos que é de sua recomendação para projetos iniciais utilizando o XP. Também se pôde observar que através dos estudos de SOUSA *et. al* [SOU2004a, SOU2004b] que os artefatos mais importantes para a documentação de software em geral estão contidos neste conjunto, com exceção do diagrama de casos de uso e do dicionário de dados, que podem ser facilmente inseridos caso o projeto em questão demonstre necessidade.

Além disso, existem esforços no sentido de se obter um processo documentativo ainda mais estruturado e disciplinado, porém sem romper com os pensamentos ágeis de desenvolvimento. Um desses esforços é o *Agile Modeling*, que por meio da combinação de práticas avançadas de modelagem e o pensamento ágil incorpora ao XP uma maior disciplina na produção de documentação essencial e de qualidade [AMB2002, BAI2002].

As experiências vividas e descritas neste trabalho demonstraram que a aplicação do XP proporciona um alto dinamismo no processo de desenvolvimento, fazendo com que a evolução do processo seja rapidamente visualizada. Além disso, o processo de modelagem e documentação quando feito levando em consideração as práticas e valores do AM, assim como a dimensão e complexidade do sistema a ser desenvolvido, proporcionam a quantidade e a qualidade das informações necessárias acerca do sistema.

Deste modo, como trabalho futuro, propõe-se a organização de uma equipe de desenvolvimento baseada em pares e seu gerenciamento, empregando-a na aplicação dos conceitos introduzidos pelo XP, assim como pelo AM, com o objetivo de se promover a efetiva avaliação de todos os aspectos práticos que envolvem estas duas metodologias associadas.

Deste modo, a definição de uma metodologia que abandona o processo de documentação tão comumente atribuída ao XP não passa de um mito, na qual o presente trabalho vem a esclarecer.

GLOSSÁRIO

Termo	Definição
API	Do inglês *Application Programming Interface*, biblioteca de funções destinadas a facilitar o desenvolvimento de aplicações.
Build	Processo de compilação e ligação do código-fonte de um software.
Code Completion	Capacidade que uma IDE pode possuir, na qual consiste em fornecer sugestões para auto-completar o código que está sendo escrito pelo desenvolvedor.
Depuração	Processo de procura e correção de erros em um software.
GUI	Do inglês *Graphical User Interface*, consiste no mecanismo de interação entre o usuário e sistema de computador baseado em símbolos visuais, como ícones, menus e janelas.
HTML	Do inglês *Hyper Text Markup Language*, trata-se de uma linguagem de marcação utilizada para produzir páginas na Internet.
IDE	Do inglês *Integrated Development Environment*, consiste em um sistema de computador que tem por objetivo integrar e disponibilizar facilidades para o desenvolvimento de softwares em uma determinada linguagem.
Navegador Web	Aplicação utilizada para a exibição de documentos na forma de páginas, dispersos na rede mundial de computadores.
Open Source	Tipo de software cujo código fonte é público.
Plug-in	Tipo de componente auxiliar que adiciona novas capacidades ou desempenham uma tarefa específica ao referido software principal.
Stakeholders	Entidades que investem em um projeto e que possuem interesse no seu sucesso.
UML	Do inglês *Unified Modeling Language*, trata-se de uma notação padronizada usada para modelar e documentar objetos de um sistema de software orientado a objetos.

REFERÊNCIAS BIBLIOGRÁFICAS

[AGI2006] Agile Alliance: Introcuction. Disponível em: <www.agilealliance.com/intro>. Acessado em 27 de setembro de 2006.

[AMB2002] AMBLER, S. W. **Agile Modeling: Effective Practices for Extreme Programming and the Unified Process**, Editora Wiley, 2002.

[BAI2002] BAIRD, S. **Teach Yourself XP in 24 Hours**, Indianapolis/USA, Editora Sams, 2002.

[BEC2000a] BECK, K. **Extreme Programming Explained: Embrace Change**. Editora Addison-Wesley, 2000.

[BEC2000b] BECK, K. **Planning Extreme Programming**. Editora Addison-Wesley, 2000

[CAR2004] CARTAXO, E. G. Testes de Aceitação em Dispositivos Móveis, Universidade Federal de Campina Grande, Novembro de 2004.

[COS1996] COSTA, R. M.; SANCHES R. Ferramentas de Engenharia Reversa no Apoio à Qualidade de Software, Relatórios Técnicos do ICMC, v. 47, 1996.

[DIA2007] Diagrama de Classes – Artigo Wikipedia: Disponível em: <http://pt.wikipedia.org/wiki/Diagrama_de_classes>. Acessado em 27 de janeiro de 2007.

[EXT2007] Extreme Modeling: Disponível em: <www.extrememodeling.com>. Acessado em 28 de janeiro de 2007.

[HAM2006] HANSSON, C.; DITTRICH, Y.; GUSTAFSSON, B.; ZARNAK, S. How agile are industrial software development practices? *The Journal of Systems and Software.* v. 79, p. 1295–1311, 2006.

[HOC2005] HOCHSTEIN, L. LINDVALL, M. Combating architectural degeneration: a survey. *Information and Software Technology*. v. 47, p. 643–656, 2005.

[HOF2002] HOFFMANN, E. Elaboração e Armazenamento de Documentação de Sistemas Informatizados, Monografia de Pós-Graduação em Gestão da Informação e Inovações Tecnológicas, FESP, Curitiba-PR, 2002.

[INS2002] INSFRÁN, E.; PELECHANO, V.; PASTOR, O. Conceptual Modeling in the eXtreme. *Information and Software Technology*. v. 44, p. 659–669, 2002.

[KET2005] KETTUNEN, P.; LAANTI, M. How to steer an embedded software project: tactics for selecting the software process model. *Information and Software Technology.* v. 47, p. 587–608, 2005.

[LAY2006] LAYMAN, L.; WILLIANS, L.; DAMIAN, D.; BURES, H. Essential communication practices for Extreme Programming in a global software

development team. *Information and Software Technology.* v. 48, p. 781–794, 2006.

[MEN2005] MENDONÇA, A.; VIEIRA, J. N.; MOURA, M. Agile Modeling, Universidade de Lisboa, Faculdade de Ciências e Tecnologia, Novembro de 2005.

[MYS2007] MySQL – Artigo Wikipedia: Disponível em: <http://pt.wikipedia.org/wiki/MySQL>. Acessado em 14 de janeiro de 2007.

[PFL2004] PFLEEGER, S. L. **Engenharia de Software: Teoria e Prática**. Segunda Edição. São Paulo: Editora Prentice Hall. 2004.

[PRE1992] PRESSMAN, R.S. **Software Engineering: A Practitioner's Approach**. 3ª ed. McGraw-Hill, 1992.

[SOM2003] SOMMERVILLE, I. **Engenharia de Software.** Sexta Edição. São Paulo: Editora Pearson Addison Wesley, 2003.

[TEL2004] TELES, V. M. **Extreme Programming**. São Paulo: Editora Novatec. 2004.

[TOF1980] TOFFLER, A. **The Third Wave**. Editora Bantam Books, 1980.

[SAK2005] SAKTHIVEL, S. Virtual workgroups in offshore systems development. *Information and Software Technology*. v. 47, p. 305–318, 2005.

[SCH2005] SCHNEIDER, J.-G.; JOHNSTON, L. eXtreme Programming – helpful or harmful in educating undergraduates? Information and Software Technology. v. 74, p. 121–132, 2005.

[SOU2004a] SOUZA, S. C. B. de; NEVES, W. C. G. das; ANQUETIL, N.; OLIVEIRA, K. M. de. Investigação da Documentação de Maior Importância para Manutenção de Software, 4ª Jornada Iberoamericana de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'04), Outubro, 2004.

[SOU2004b] SOUZA, S. C. B. de; NEVES, W. C. G. das; ANQUETIL, N.; OLIVEIRA, K. M. de. Investigação da Documentação de Maior Importância para Manutenção de Software II, XVIII Simpósio Brasileiro de Engenharia de Software (SBES'04), Outubro, 2004.

[TAY1911] TAYLOR, F. W. The Principles of Scientific Management, 1911.