Resumo

O método *Extreme Programming* (XP) caracteriza-se por ser um método incremental de desenvolvimento de software, segmentando o processo de desenvolvimento em iterações. No início destas iterações, o cliente explicita as funcionalidades, ou histórias de usuário, desejadas para o sistema. Estas histórias servem como lembretes de funcionalidades e para criar estimativas de tempo para suas implementações, além de servir como base para a criação de testes de aceitação. Após decidir a história a ser implementada durante a iteração, são realizadas discussões que detalham melhor a solicitação do cliente. Contudo, histórias não são utilizadas para geração de um modelo abstrato que descreva as funcionalidades, o que poderia trazer benefícios como aumento de produtividade e eficiência na comunicação com o cliente. Este trabalho objetiva a criação de um protótipo de ferramenta que possibilite o registro dessas histórias e de seus detalhes, tornando possível a geração automática de um diagrama de classes UML. A utilização desta ferramenta, apesar de incluir mais documentação ao processo, possibilita a utilização do modelo gerado para análise e discussão com o cliente durante a fase de codificação e a geração semi-automática de código através de uma ferramenta dirigida por modelos (MDA). A utilização de alguns elementos da notação *Business Object Notation* (BON) e a estruturação da linguagem de descrição dos detalhes das histórias permite-nos distinguir as classes, associações entre elas, atributos e métodos para a geração deste modelo abstrato que descreve o sistema.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

XP – *Extreme Programming*
UML – *Unified Modeling Language*
MDA – *Model Driven Architecture*
RUP – *Rational Unified Process*
OCL – *Object Constraint Rules*
MOF – *Meta Object Facility*
CWW – *Common Warehouse Metamodel*
OMG – *Object Managment Group*
3GLs – Linguagem de Terceira Geração
PIM – *Platform Independent Model*
PSM – *Platform Specific Model*
BON – *Business Object Notation*
XML – *Extensible Markup Language*
XSLT – XML *StyleSheet Language Transformation*
W3C – *World Wide Web Consortium*

Agradecimentos

Agradeço principalmente a Deus que me proporcionou oportunidades e me conduziu desde o berço até este momento mágico onde passo a ser reconhecido como profissional. Posso sintetizar tudo isso em uma das promessas de Jesus Cristo: “Buscai primeiro o reino de Deus e tudo vos será acrescentado”. Foi realmente isso o que aconteceu! Deus mostrava os caminho e eu neles seguia, claro, com as lutas que eram necessárias.

Agradeço também a minha família que sempre apoiou minhas decisões e, em especial, dedico esta vitória a meu Pai Agenor, o maior incentivador dos meus estudos e um verdadeiro guerreiro que, mesmo tendo passado momentos difíceis, nunca deixou que nos faltasse nada. Ao seu lado, sempre estava minha mãe Rosineide e juntos nos ouviam e aconselhavam, além de nos ter transferido valores que valem mais que qualquer riqueza material: meus pais são um presente de Deus.

Também agradeço a minha namorada Leila, que nesta correria contra o tempo, mostrou-se compreensiva e sempre apoiou as minhas atividades profissionais mesmo tendo que abrir mão da minha presença em vários momentos. Você também é um presente para mim.

Muito obrigado também ao meu amigo Rodrigo Lobo que, acreditando em mim, abriu as portas para uma oportunidade de estágio que me fez e faz aprender muito, contribuindo assim com este trabalho de conclusão de curso. Ao meu orientador Dr. Márcio Lopes Cornélio, também meus agradecimentos. Este trabalho só foi possível graças a sua criatividade e competência. Ah! Sua paciência também foi valiosa...

Não poderia deixar de agradecer a todos os professores do Departamento de Sistemas Computacionais da UPE, em especial, o professor Dr. Fernando Buarque que sempre nos provocou com questionamentos que me ajudaram a ser o profissional e o homem que sou hoje.

Por fim meu muito obrigado a todos os companheiros que fizeram parte da minha vida nesses últimos 5 anos. Aprendi um pouco com cada um e realmente sentirei falta da nossa convivência diária.

Deus abençoe a todos e muito obrigado!

Capítulo 1

Introdução

Processo de desenvolvimento de software é um termo referente a um conjunto de atividades e resultados associados que produzem um produto de software[1]. Sua definição contempla procedimentos e métodos, ferramentas e equipamentos, comunicação e pessoas. Na prática, estabelece a organização das atividades a fim de garantir o desenvolvimento do produto de software e a sua qualidade.

A criação destes processos e posteriores melhorias surgiram após o período conhecido como a *crise do software* ocorrida na década de 1960, onde vários projetos excediam a previsão de cronograma e custos, além de apresentarem baixa qualidade.

O mais conhecido e antigo processo desta natureza é o processo linear e seqüencial de desenvolvimento, também conhecido como cascata. Ele divide o desenvolvimento em quatro grandes etapas realizadas seqüencialmente: análise, projeto, codificação e testes. Este modelo ainda é amplamente utilizado hoje[2].

A utilização deste processo seqüencial baseado na criação de artefatos em cada fase é uma tentativa de disciplinar o desenvolvimento e buscar os níveis de previsibilidade desejados. Contudo, freqüentemente se obtém apenas formalismos e não disciplina. Além disso, esse modelo não provê mecanismos de *feedback* do cliente, o que aumenta os riscos de que o projeto entregue não satisfaça as suas necessidades.

Em meados da década 80 foi proposto o modelo em espiral, que presume que o processo de desenvolvimento ocorre em ciclos, cada um contendo fases de avaliação e planejamento, onde a opção de abordagem para a próxima fase (ou ciclo) é determinada, podendo acomodar características de outros modelos. O modelo original em espiral organiza o desenvolvimento como um processo iterativo em que vários conjuntos de quatro fases se sucedem até se obter o sistema final[2].

Ao longo do tempo, a indústria de software criou mais alternativas para os processos de desenvolvimento de software. O *Rational Unified Process* (RUP), criado na década de 90, é um

framework de processo bastante abrangente que, como tal, pode ser instanciado para atender às necessidades dos mais diversos tipos de projetos de software[3].

O RUP sugere um conjunto de boas práticas derivadas a partir de projetos de desenvolvimento bem-sucedidos e da análise de erros em projetos mal-sucedidos. Além disso, este modelo estabelece o desenvolvimento iterativo e incremental como forma de incorporar *feedback* e aprendizado ao processo. Porém, ainda torna o desenvolvimento burocrático.

Na década de 1990, alguns profissionais da indústria de software começaram a propor novos processos de desenvolvimento menos burocráticos e mais adequados para lidar com aspectos humanos dos projetos de software. Em 2001, dezessete profissionais experientes se reuniram nos Estados Unidos a fim de discutirem as bases comuns às suas propostas de metodologia, o que ficou conhecido como *manifesto ágil*[4].

O Programação Extrema (XP) é o método ágil mais difundido. Ele sugere um conjunto de valores, princípios e práticas que visam a satisfação do cliente. Adequação a variações no escopo do projeto e ciclos de *feedbacks* são algumas das características do XP, garantidas por mecanismos de proteção do projeto como a adoção de iterações curtas e a presença do cliente, ou um de seus representantes, no ambiente de desenvolvimento[5].

O início de cada iteração XP é marcado pela escolha do cliente acerca de funcionalidades desejadas, documentadas em cartões conhecidos como cartões de história ou histórias de usuário[5], que contêm breves descrições das funcionalidades. É realizado um trabalho conjunto entre cliente e equipe de desenvolvimento para escolha de algumas delas, a fim de serem implementadas durante a iteração atual.

Histórias servem apenas como lembretes de funcionalidade e seus detalhes são melhor entendidos no início da codificação quando o desenvolvedor, tendo acesso direto ao cliente, o interroga sobre suas necessidades ao mesmo tempo em que aprende sobre o negócio a que o sistema está relacionado.

Este trabalho visa à construção de um protótipo de ferramenta para cadastro e gerenciamento de histórias de usuário, utilizando-as para geração automática de um diagrama de classes definido pela *Unified Modeling Language* (UML)[6].

Neste capítulo apresentaremos os objetivos do trabalho e sua justificativa. No decorrer do documento detalharemos como foi implementada a ferramenta proposta.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver um protótipo de ferramenta a ser utilizada por equipes de desenvolvimento cujo processo de construção de software englobe práticas do método ágil XP, a fim de aumentar a produtividade e eficiência dessas equipes. Tal ferramenta visa o registro de histórias de usuário e seus detalhes para que, de forma automática, possamos construir um diagrama de classes. O modelo abstrato sempre conterá as funcionalidades a serem implementadas na corrente iteração do XP.

Apresentaremos os detalhes de sua implementação, discutindo os aspectos técnicos e o ponto de inserção da ferramenta no processo de desenvolvimento.

Também objetivamos realizar uma prova de conceito onde aplicaremos a ferramenta desenvolvida às iterações iniciais de um sistema,. Após a obtenção do modelo abstrato, utilizaremos uma ferramenta de geração de código baseada em modelos[7], a exemplo do BaseGen[8] ou ArgoUML[9], a fim de obtermos a implementação de algumas classes do sistema.

1.2 Justificativa

O XP, como um método ágil de desenvolvimento, sugere uma série de práticas e valores a serem adotados no processo de construção do software. Estes buscam, entre outros objetivos, a satisfação do cliente e a diminuição de artefatos e documentação gerados durante o processo[5].

Contudo, acreditamos que o registro das histórias e seus detalhes, além de constituírem uma documentação útil ao longo do processo de desenvolvimento, permitirá o aumento da produtividade e eficiência das equipes que adotarem a ferramenta para o planejamento da iteração no método XP. Isto porque o modelo abstrato gerado poderá ser utilizado para construção de classes de forma semi-automática além de constituir um elemento útil para a discussão entre cliente e desenvolvedores durante a fase de codificação.

Dentre outros benefícios, podemos citar a possibilidade de geração automática de documentação do sistema, uma vez que as informações de comportamento do mesmo estarão registradas como detalhamento das histórias. A geração automática de classes de testes também constitui um possível benefício de utilização da ferramenta. Contudo, este trabalho não visa implementar estas duas funcionalidades.

1.3 Estrutura do Trabalho

Este trabalho de conclusão de curso está estruturado de acordo com a seguinte lista de capítulos:

- No Capítulo 2, abordaremos os conceitos de métodos ágeis, focalizando nos valores e princípios do XP. Além disso, discutiremos aspectos relacionados à geração de código a partir de modelos e os aspectos da notação *Business Object Notation* nas quais este trabalho foi inspirado. Os tópicos restantes trazem informações sobre a transformação de documentos XML com a utilização de XSLT e a representação de modelos UML com a utilização de XMI.
- No Capítulo 3, descreveremos decisões de projeto da ferramenta, mostrando os passos necessários para a geração do diagrama de classes, e os aspectos técnicos de sua implementação. Além disso, dissertaremos sobre o ponto de inserção da ferramenta no processo XP e abordaremos as limitações desta ferramenta.

- No Capítulo 4, simularemos a primeira iteração XP para o desenvolvimento de um sistema, a fim de realizarmos uma prova de conceito. Utilizaremos a ferramenta para geração do diagrama de classes e importaremos o modelo gerado com uma ferramenta de modelagem, a partir do qual geraremos código-fonte para o modelo.
- No Capítulo 5, concluiremos a pesquisa dissertando sobre as contribuições alcançadas e direções futuras de trabalhos.

Capítulo 2

Fundamentação Teórica

Este capítulo aborda o conceito de metodologia ágil e uma breve comparação com metodologias mais tradicionais como o RUP. O XP é o método ágil detalhado para exemplificar os conceitos, sobretudo porque é o método ao qual a contribuição apresentada neste trabalho se aplica, mas também porque é o método ágil mais popular.

O entendimento dos princípios e valores do XP é de fundamental importância para analisar o contexto em que se insere nossa contribuição e, ao mesmo tempo, revela-se essencial para entender os seus benefícios.

2.1 Métodos Ágeis

Existem muitos métodos de desenvolvimento de software nesta categoria de métodos, como XP, SCRUM[10] e Crystal[11]. Mas a grande parte dos métodos ágeis segmenta o processo de desenvolvimento em iterações, a fim de desenvolver miniaturas do projeto ao invés de manipulá-lo como um todo. Iterações são intervalos de tempo pré-definidos para realização de um subconjunto das tarefas do projeto e, ao final de cada uma dessas iterações, um projeto de desenvolvimento ágil visa implantar uma nova versão do software. Para isso são realizadas algumas tarefas no decorrer das mesmas: planejamento, análise dos requisitos, projeto, codificação, teste e documentação. Ainda no final de cada iteração, a equipe envolvida avalia os pontos positivos e reavalia as prioridades do projeto.

Projetos ágeis apresentam um conjunto de valores estabelecidos pelo *manifesto ágil*, citado no capítulo 1 deste trabalho:

- **Indivíduos e Iterações** ao invés de processos e ferramentas
- **Software funcionando** ao invés de documentação abrangente
- **Colaboração com o cliente** ao invés de negociação de contratos
- **Responder as mudanças** ao invés de seguir um plano

Os métodos ágeis enfatizam comunicação em tempo real ao invés de documentos escritos. Em termos práticos os componentes de uma equipe de desenvolvimento e o cliente dividem um mesmo ambiente, ou seja, os desenvolvedores, gerentes, testadores, projetistas da iteração e o cliente (ou um de seus representantes), comunicam-se diretamente assim que necessitarem.

O método adotado em um processo de desenvolvimento de software consiste em fator importante na determinação da probabilidade de sucesso do projeto[10]. Por isso, métodos ágeis vêm sendo amplamente utilizados para guiar o processo de desenvolvimento de sistemas computacionais, sobretudo porque aumentam a produtividade e reduzem os custos organizacionais, mas principalmente porque provêem adaptabilidade às mudanças naturais no escopo do projeto[12].

2.1.1 *Extreme Programming* (XP)

Esta subseção foi baseada no texto de Beck. Kent[5] e discute as principais características do método de desenvolvimento XP. Entre os métodos de desenvolvimento de natureza ágil, este é o que mais se destaca. Ele apresenta uma série de práticas que objetivam principalmente a satisfação do cliente.

XP está estruturado sobre valores, a partir dos quais se criou princípios, que por sua vez direcionam as práticas do XP. A metodologia é flexível e prevê liberdade de criação: se a equipe necessita de práticas adicionais não previstas ainda, a própria equipe pode criá-las tendo por base os princípios e valores.

Os principais fundamentos do XP tiveram origem nas tradições do desenvolvimento em *Smalltalk* e datam de meados da década de 80, quando Kent Beck e Ward Cunningham trabalhavam na *Tektronixs, Inc.* Práticas, tais como, *refactoring*, programação em pares, mudanças rápidas, *feedback* constante do cliente, desenvolvimento iterativo, testes automatizados, entre outras, são elementos centrais da cultura da comunidade *Smalltalk*. Olhando deste ponto de vista, XP pode ser considerado o modo de agir deta comunidade generalizado para outros ambientes.

Segundo Kent Beck,[5] o XP difere de outras metodologias por[5]:

- Apresentar *feedback* (retornos) contínuos e concretos em ciclos curtos;
- Abordar planejamento incremental, apresentando rapidamente um plano global, que evolui durante o ciclo de vida do projeto;
- Ter habilidade flexível de programar implementação de funcionalidade, respondendo às mudanças das regras de negócio;
- Confiar nos testes automatizados escritos pelos programadores e clientes para monitorar o progresso do desenvolvimento, permitindo a evolução do sistema e detectando antecipadamente os problemas;
- Acreditar na comunicação oral, na colaboração íntima dos programadores, nos testes e no código fonte, definindo a estrutura do sistema e os objetivos;
- Confiar no processo de evolução do projeto, que dura tanto quanto o sistema;
- Acreditar nas práticas que trabalham tanto com as aptidões, em curto prazo, dos programadores, quanto os interesses, em longo prazo, do projeto.

2.1.2 Valores do XP

É de fundamental importância ressaltar que o XP segue uma série de valores, princípios e regras básicas que visam alcançar eficiência e satisfação no processo de desenvolvimento de software[5].

Os quatro valores do XP são:

- **Comunicação:**

  O desenvolvimento de software é uma tarefa que envolve, no mínimo, duas pessoas: o usuário e o desenvolvedor. Projetos reais geralmente possuem mais pessoas envolvidas e a comunicação constitui um valor essencial para o projeto.

  Projetos XP procuram envolver ativamente seus usuários (ou ao menos um representante dos mesmos), tornando-os integrantes da equipe de desenvolvimento. Mais especificamente, equipe de desenvolvimento e o usuário (ou seu representante) trabalham no mesmo local, possibilitando à equipe, acesso rápido a um especialista no domínio do negócio. Isso acelera o fluxo de informações e torna a comunicação baseada principalmente em diálogos presenciais, ao invés de telefonemas ou e-mails.

- **Simplicidade:**

  Em métodos tradicionais o escopo do projeto é definido no início do mesmo e alterações neste escopo são evitadas ao longo do processo. A fim de evitar tais alterações indesejadas, a equipe de desenvolvimento tenta antecipar possíveis solicitações futuras do usuário e criar soluções genéricas para "facilitar" possíveis alterações no escopo. Contudo, em diversos projetos, estas atitudes da equipe causam um trabalho desnecessário, aumentando os custos e provocando atrasos. O XP recomenda que as funcionalidades projetadas para cada iteração sejam implementadas com a maior qualidade possível, mas focalizando apenas os aspectos essenciais. O desenvolvedor deve procurar simplificar o sistema, e assim tornar mais fácil possíveis alterações futuras, ao invés de tentar prever funcionalidades que o usuário poderá solicitar.

- ***Feedback***

  Este valor está intimamente ligado ao entendimento do sistema pela equipe de desenvolvimento. As tarefas de transmitir as necessidades do usuário e de entendê-las são muito críticas e importantes, pois determinam os demais esforços na construção do software. Na prática, os usuários não têm como prever corretamente as funcionalidades que necessitarão, motivo que torna fundamental a interação com os desenvolvedores ao longo do projeto. A compreensão das necessidades do usuário é um processo de aprendizado contínuo no qual os desenvolvedores aprendem sobre os problemas do negócio e os usuários tomam conhecimento das dificuldades e limitações técnicas.

- **Coragem**

  O processo de desenvolvimento pode acarretar alguns temores para usuários e desenvolvedores [13]:

  - Usuários podem fazer solicitações erradas, não obter o que pediram, ter grandes gastos por poucas coisas ou desconhecer os acontecimentos do projeto.
  - Desenvolvedores podem ser solicitados a fazer mais do que sabem, receber responsabilidades sem autoridade necessária para honrá-las, não obter informações claras e suficientes sobre o que precisa ser realizado, e terem que sacrificar a qualidade em função do prazo.

Coragem em XP refere-se à confiança nos mecanismos de segurança utilizados para proteger o projeto. Os temores citados são reconhecidos nos projetos de XP como problemas que irão ocorrer, mas a utilização desses mecanismos de proteção ajuda a reduzir ou eliminar as conseqüências desses problemas.

Algumas das características do XP como a iteratividade, e a adoção de iterações curtas e fixas, ajuda na proteção do cliente, por exemplo. Ao final de cada iteração, é possível avaliar se a equipe implementou o que foi solicitado e se o que foi solicitado realmente fazia sentido.

Ressaltamos, porém que a iteratividade não resolve o problema, mas permite seu diagnóstico de forma mais rápida, o que facilita eventuais correções e evita que a equipe invista muitos recursos em funcionalidades incorretas, caso o cliente tenha errado ao solicitá-las[14].

2.1.3 Práticas do XP

Práticas do XP dizem respeito ao modo de agir das equipes de desenvolvimento. Na verdade, estas práticas são definidas sobre os valores e princípios do método.

Cada prática traz benefícios para o processo, contudo esses benefícios obtidos são extremamente maiores quando essas práticas são combinadas entre si[15]. Tais práticas são divididas em dois grupos: práticas primárias e corolárias[15]. Estas últimas não devem ser adotadas até um bom domínio das práticas primárias. Todavia as práticas primárias são o conjunto de práticas recomendadas que devem ser adotadas por qualquer equipe com qualquer grau de experiência no método.

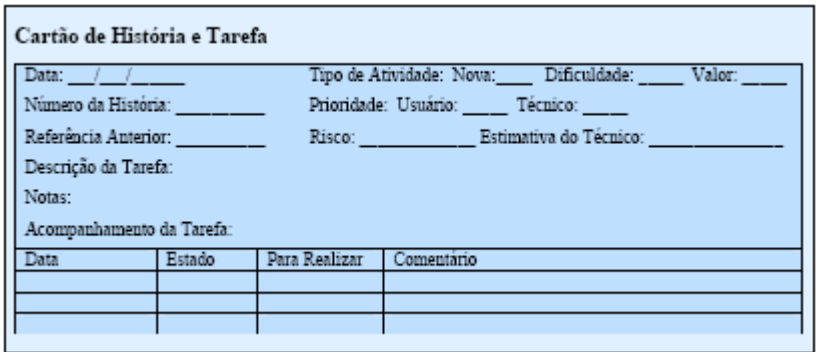
A seguir encontram-se algumas das principais práticas primárias e corolárias sugeridas pelo XP:

- **Jogo do Planejamento (*Planning Game*):** Planejamento em XP diz respeito aos rumos que o projeto irá tomar ao invés de especificar exatamente o que será necessário e quanto levará para seu término. Basicamente duas questões são trabalhadas: (i) predizer o que será realizado até determinada data e (ii) determinar qual a próxima tarefa a ser executada. Existem dois passos chave em planejamento XP, abordando essas duas questões:

  Planejamento de Versão (*Release Planning*) é o passo onde o cliente apresenta as funcionalidades desejadas para a iteração. Estas funcionalidades são escritas pelos clientes em cerca de três linhas de texto sem termos técnicos e servem apenas para criar estimativas de tempo para a funcionalidade além de servir como base para a criação de testes de aceitação.

  Tais funcionalidades são dispostas em cartões e são chamadas de histórias de usuário ou cartão de história. A Figura 1 exibe a estrutura de um cartão de história.

  Planejamento de Iteração (*Iteration Planning*) diz respeito ao planejamento da próxima iteração. Dadas as funcionalidades requisitadas pelo cliente no planejamento de versão, os desenvolvedores as dividem em tarefas e as estimam com um maior nível de detalhe em relação à estimativa de planejamento de versão. Então, baseados no volume de trabalho realizado na iteração prévia, estimam o tempo necessário para iteração corrente.



Cartão de História e Tarefa

Data: __/__/_____ Tipo de Atividade: Nova:____ Dificuldade: _____ Valor: _____
Número da História: _____ Prioridade: Usuário: _____ Técnico: _____
Referência Anterior: _____ Risco: _____ Estimativa do Técnico: _____
Descrição da Tarefa:
Notas:
Acompanhamento da Tarefa:

Data	Estado	Para Realizar	Comentário

Figura 1. Modelo de Cartão de História (Adaptado de [5]).

- **Pequenas Versões (*Small Releases*):** Ao final de cada iteração, a equipe de desenvolvimento disponibiliza um executável do sistema com as funcionalidades implementadas durante a iteração corrente. O cliente pode utilizar este software para qualquer propósito, tanto para avaliação, quanto para liberar para os usuários finais.

- **Projeto Simples:** As iterações em XP geralmente são um intervalo de tempo entre duas e três semanas. Assim, torna-se necessário que a equipe de desenvolvimento mantenha o sistema com um projeto suficientemente simples. Iterações devem ser oportunidades para que a arquitetura seja revisada e aprimorada caso necessário.

- **Teste:** Os programadores primeiramente escrevem os testes de unidade e o cliente escreve os testes de aceitação, de forma que a confiança na funcionalidade possa se tornar parte do programa desenvolvido[5]. É importante ressaltar que os testes são escritos antes da implementação.

- ***Refactoring*:** Processo contínuo de melhoria de projeto para garantir que o software estará bem projetado. Na prática, *refactoring* é uma técnica empregada na reestruturação do código, cujo principal propósito é fazer com que programa fique mais reutilizável e fácil de entender, sem que haja mudança de comportamento[16].

- **Programação em Pares (*Pair Programming*):** Esta é uma prática que visa melhorar a qualidade do código e testes escritos através do trabalho conjunto de dois desenvolvedores que se unem para implementar, juntos, as funcionalidades da iteração. Esta técnica implementa uma das diversas redes de proteção que os projetos XP utilizam para reduzir os riscos de eventuais falhas.

- **Propriedade Coletiva (*Colletive Ownership*):** O time XP como um todo é responsável por todo código escrito. Desta maneira, não é necessário pedir autorização para realizar alterações em qualquer programa no repositório. Esta prática também protege o projeto ajudando a tornar a equipe mais robusta, na medida em que os desenvolvedores se habituam a trabalhar nas mais variadas partes do sistema.

- **Cliente dedicado:** O cliente deve ter papel ativo no processo de desenvolvimento, sobretudo porque é um dos principais interessados no projeto, mas também porque possui o conhecimento de negócio. Assim, o XP incorpora o cliente à equipe de desenvolvimento, o que significa colocar o cliente fisicamente próximo aos desenvolvedores ou mover os desenvolvedores para perto do cliente. A presença do cliente ao longo do desenvolvimento viabiliza o ciclo contínuo de *feedback* entre ele e os desenvolvedores. Este ciclo permite que pequenas mudanças sejam feitas ao longo do desenvolvimento, de forma rápida.

2.1.4 Ciclo de Vida e Fases do Processo XP

Um projeto XP ideal começa com um desenvolvimento curto de funcionalidades, seguido de anos de produção e refinamentos simultâneos até que o projeto não faça mais sentido e finalmente acabe[5].

O ciclo de vida do XP costuma ser curto e esta abordagem faz sentido apenas em ambientes onde as mudanças de requisitos do sistema são freqüentes. A seguir detalhamos as fases do processo XP:

2.1.4.1 Exploração

Esta fase objetiva o entendimento sobre o que o sistema deve fazer, bem o suficiente para que possa ser estimado. Para isso, o cliente escreve e administra cartões de histórias e o programador as estima. Esta fase de exploração se encerra quando há histórias suficientes para a próxima fase.

Esta mesma fase deve proporcionar segurança suficiente para que a equipe de desenvolvimento acredite que tem conhecimento, ou que pode buscá-lo, a fim de iniciar e finalizar o projeto. Enfim, a equipe deve aprender a confiar em todos os membros do time XP[5].

Como explicado, esta fase se inicia com a escrita de histórias que descrevem o que o sistema de fazer. Tais histórias devem apresentar alguns atributos para que os objetivos de sua existência, como a estimação, sejam alcançados. Uma história deve ser[17]:

- **Independente:** Deve-se evitar a introdução de dependências entre as histórias, pois isto causa problemas em suas priorizações e planejamento, além de tornar as estimativas mais difíceis.
- **Negociável:** Histórias não devem conter muita informação sobre o requisito, pois elas servem como lembretes para que os detalhes relevantes sejam discutidos posteriormente.
- **Valiosa para usuários e clientes:** Na maioria dos sistemas existe uma distinção entre usuário (aquele que utiliza o sistema) e clientes (aquele que compra o sistema). Histórias também devem ter valor para os usuários.
- **Estimável:** Desenvolvedores devem poder estimar o tamanho das histórias ou a quantidade de tempo necessária para sua codificação. Entre os fatores que impedem isso, destacam-se a falta de conhecimento de negócio e técnico dos desenvolvedores, assim como o tamanho da história.
- **Pequena:** Histórias devem ser suficientemente pequenas para que sirvam de lembrete sobre a funcionalidade, sejam estimáveis e se possa projetar formas de testá-las. Uma história grande pode ser dividida em outras menores.
- **Testável:** O XP enfatiza o projeto de testes antes mesmo da codificação. Desta forma, histórias devem conter informações que permitam a criação dos mesmos. Testes são a

forma de verificar se a funcionalidade foi corretamente implementada, daí sua importância.

Em paralelo com as histórias, os desenvolvedores vão explorando as possibilidades de arquitetura para o sistema e verificam a tecnologia de implementação. A exploração de arquitetura dura cerca de duas semanas e isso auxilia os desenvolvedores quando o usuário apresentar seus cartões de histórias[5].

2.1.4.2 Planejamento

Após estimar as histórias, dá-se início à fase de planejamento que visa definir a menor data e o maior conjunto de histórias que serão realizadas na primeira versão[5].

O jogo do planejamento, como já foi citado, é a melhor maneira de executar esta fase. O cliente decide quais histórias são vitais e devem ser implementadas na primeira versão. Desta forma, elabora-se uma lista priorizada de histórias.

Alguns passos, que também podem ser visualizados na Figura 2, são apresentados para auxiliar a fase de *release* no jogo do planejamento[18]:

- O cliente seleciona histórias baseados em seus valores: alto, médio ou baixo
- Os desenvolvedores classificam as histórias por risco (opcional): alto, médio ou baixo.
- Os desenvolvedores declaram a velocidade: calculada sobre a estimativa realizada sobre as histórias dos clientes. A velocidade é empiricamente determinada, ou seja, baseada na experiência dos desenvolvedores.
- Cliente escolhe o escopo: escolhe histórias para a próxima versão.

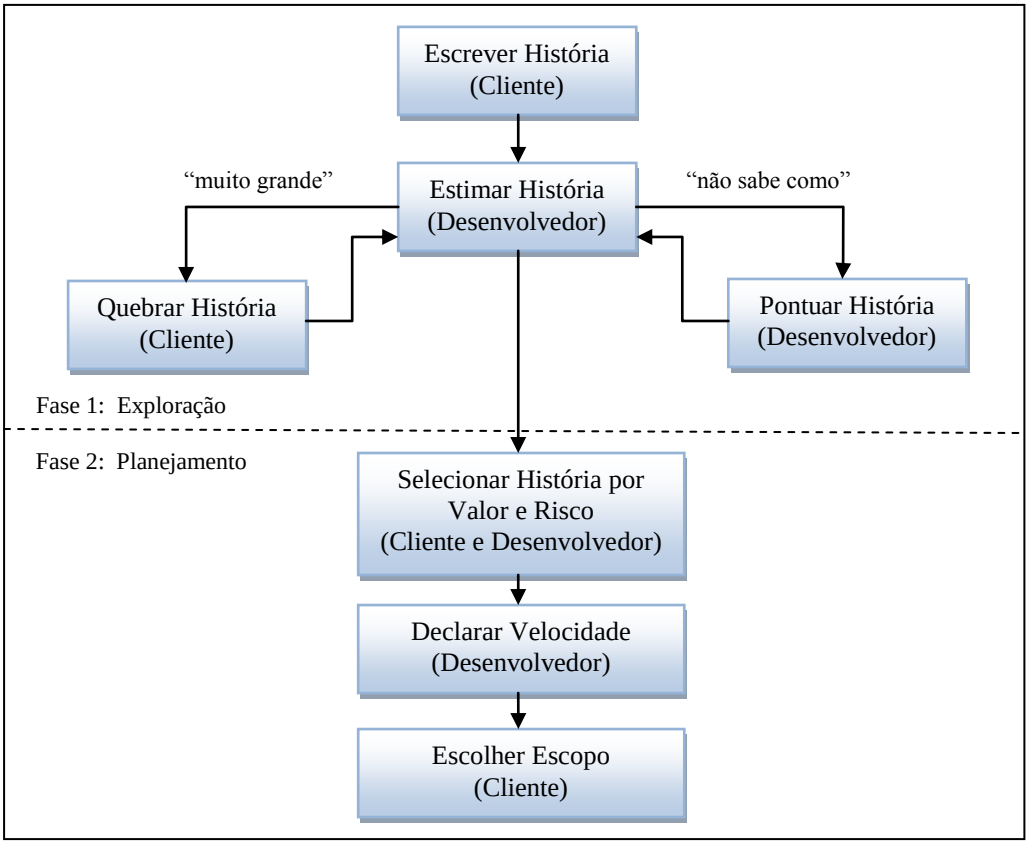


Figura 2. Jogo do Planejamento: *Release*

No primeiro dia de cada iteração a equipe de desenvolvimento lê os cartões de histórias e as quebra em pequenas tarefas (*task cards*) e então são definidas atribuições para cada desenvolvedor.

2.1.4.3 Iteração para a primeira versão

Os compromissos são divididos para serem executados em iterações que duram de uma a quatro semanas[5]. Para cada história executada naquela iteração é produzido um conjunto de testes funcionais.

A primeira iteração mostra como a arquitetura do sistema irá se comportar[5]. Então, as histórias devem ser escolhidas de forma que representem força para criar todo o sistema. A pergunta chave para ser trabalhada nesta fase é: qual a coisa de maior valor para o time para ser trabalhada nesta iteração?[5]

2.1.4.4 Produção

Alguns processos podem ser utilizados para avaliar se o software realmente está pronto para entrar em produção[5]. Testes são freqüentemente aplicados nesta fase a fim de verificar sua estabilidade.

2.1.4.5 Manutenção

Este é o estado normal de um projeto XP. Nele, deve-se simultaneamente produzir novas funcionalidades, manter o sistema existente em produção, substituir membros do time que partem e incorporar novos membros ao time[5].

Também nesta fase deve-se realizar atividades de *refactoring*, a fim de melhorar a estrutura de código e, se necessário, aumentar a sua manutenibilidade.

2.1.4.6 Fim do Projeto

O fim do projeto ocorre com a falta de novas histórias. Este é o momento de escrever algumas páginas sobre a funcionalidade do sistema, um documento que auxilie, no futuro, a saber como realizar uma alteração no sistema. Uma boa razão para finalizar o sistema é o cliente estar satisfeito com o sistema e não ter mais nada que consiga prever para o futuro.

2.2 *Model Driven Architecture* (MDA)

Arquitetura dirigida por modelos (MDA)[7] é uma abordagem para especificação de sistemas que separa a especificação de funcionalidade da especificação da implementação desta funcionalidade em uma plataforma específica. Tal abordagem é proposta e financiada pela *Object Managment Group* (OMG)[19].

A MDA e os padrões que dão suporte a ela permitem que o mesmo modelo seja concretizado em múltiplas plataformas através de padrões de mapeamento auxiliares, ou através de mapeamentos para plataformas específicas, além de permitir que diferentes aplicações sejam integradas ao relacionar explicitamente os seus modelos, garantindo assim integração e interoperabilidade.

Além desses benefícios, há outro muito importante. No começo dos anos 60 a indústria estava no meio de uma revolução. O uso de linguagens de montagem foi substituído pelo uso de linguagens procedimentais, ou linguagens de terceira geração (3GLs) como eram chamadas. Programas escritos em linguagens procedimentais eram compilados, ou transformados, para programas escritos em *assembly*. Na época, os compiladores não estavam maduros o suficiente. O código compilado apresentava muitas vezes perda de desempenho e eficiência com relação ao código *assembly*. Contudo, as vantagens das linguagens de alto nível se tornaram óbvias fazendo com que a sua utilização se consolidasse com o tempo.

A MDA, atualmente, está nos trazendo outra mudança. A automação de software através de modelos é um outro nível de compilação. Isto é, modelos são compilados para 3GLs, que são compilados para linguagens *assembly*. Os compiladores MDA (leia-se ferramentas de transformação) ainda não estão maduros. Contudo, a vantagem de trabalhar em um nível mais alto de abstração será óbvia da mesma forma que o uso de 3GLs foi. Esta mudança, apesar de demorar certo tempo para se concretizar, faz com que MDA possua um grande potencial para melhorar de forma significativa o processo de desenvolvimento de software[20].

Os principais benefícios da MDA, de acordo com a OMG, são[7]:

- **Produtividade:** Os desenvolvedores não precisam se preocupar com detalhes específicos da plataforma-alvo. Detalhes técnicos específicos são implementados diretamente no *Platform Specific Model* (PSM)[7], relacionado à plataforma. Além disso, a maior parte do código também é gerada de forma automática com a transformação do PSM em código-fonte.
- **Portabilidade:** A portabilidade se encontra no *Platform Independent Model* (PIM)[7] que pode ser estendido para qualquer plataforma através de um PSM que a represente.
- **Interoperabilidade:** A interoperabilidade a que o MDA se refere diz respeito a possibilidade de utilização de um PIM para comunicação entre PSMs.
- **Manutenção e Documentação:** Com a ajuda de ferramentas MDA, ao se realizar alguma alteração em um PSM, esta também se reflete sobre o PIM de origem, mantendo PIM, PSM e documentação consistentes até o término do projeto.

Simplificadamente, MDA pode ser descrito como uma visão de como o software pode ser desenvolvido colocando o modelo UML[6] no centro do processo de desenvolvimento. A partir de um modelo abstrato do sistema é gerado então um modelo mais concreto. A partir deste último modelo, o código-fonte pode ser derivado. A idéia deste processo é automatizar ao máximo cada etapa da geração. Esse conjunto de transformações faz parte do processo MDA, que detalharemos mais adiante.

Como foi afirmado, o MDA consiste na utilização de padrões independentes de plataforma. Entre esses padrões destacamos:

- **UML (*Unified Modeling Language*)**: Linguagem padrão para descrever modelos orientados a objeto[6].
- **OCL (*Object Constraint Language*):** Linguagem que torna os modelos UML mais precisos, definindo restrições e tornando-os consistentes e não ambíguos.

- **XMI (*XML Metadata Interchange*):** Padrão que define formato de dados para permitir intercâmbio de modelos UML[21].
- **MOF (*Meta Object Facility*):** Padrão que define como meta-modelos devem ser estruturados.
- **CWM (*Common Warehouse Metamodel*):** Especificação que descreve intercâmbio de *metadados* entre *data warehousing, business intelligence,* gerência de conhecimento e tecnologias de portal.

2.2.1 O processo MDA

O processo MDA consiste nos passos necessários para através de um modelo abstrato do sistema obtermos código em uma plataforma específica através de algumas transformações.

O primeiro passo deste processo consiste na criação de um modelo independente de plataforma (PIM ou *Platform Independent Model*). Então a partir deste PIM é gerado um modelo específico de plataforma (PSM ou *Platform Specific Model*). Em termos práticos o PSM é parecido com o PIM, mas inclui anotações especiais dependentes da tecnologia subjacente. Após isso, o PSM deve ser transformado em código.

A separação entre o PIM e o PSM traz alguns importantes benefícios[20]:

- A validação e correção de um modelo PIM são simplificados já que ele não contempla aspectos semânticos da plataforma.
- A extensão do modelo para uma nova plataforma, mantendo a mesma estrutura e comportamento do sistema, é facilitada.
- A integração e interoperabilidade entre modelos são mais facilmente definidos sobre os modelos PIM.

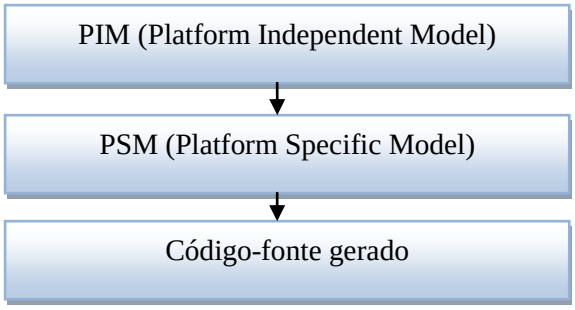


Figura 3. As etapas do processo MDA

O PIM é o centro do *framework* MDA. Isto torna a modelagem, de fato, uma programação em um nível mais alto[8]. O PIM especifica o código que precisa ser produzido. Haverá bem menos codificação feita à mão. Desta forma, o esforço de implementação se resume à criação de um modelo completo do sistema que seja bem anotado, independente de tecnologia e de alto nível.

2.2.2 Componentes de um framework MDA

Os componentes necessários para a implementação do processo MDA são os seguintes[8]:

- Um ou mais padrões de linguagens, a exemplo de UML e OCL[22], para escrever modelos de alto nível.
- Modelos de alto nível, escritos nas linguagens escolhidas, de forma consistente e precisa, e que contenham informações suficientes sobre o sistema.
- Definição de como um PIM é transformado em um PSM específico que possa ser executado automaticamente.
- Uma linguagem para descrever as definições de transformação. Essa linguagem deve ser utilizada pelas ferramentas de transformação.
- Ferramentas que implementem a execução das definições de transformação (PIM para PSM).
- Ferramentas que implementem a execução da transformação do PSM para o código.

Como afirmado anteriormente, o MDA coloca a responsabilidade do sistema no PIM. Por isso, este modelo deve ser completo e não um simples diagrama e alguns textos. O PIM deve ser mais formal, preciso, consistente, e conter o máximo de informações sobre o sistema.

2.3 *Business Object Notation* (BON)

Esta seção apresenta uma breve descrição dos principais conceitos da notação BON[23] e foi baseada no texto de Wálden, Kim e Nerson, J.[23]. Este trabalho apenas se inspira em alguns dos conceito desta notação.

BON pode ser visto como um método com um conjunto de regras e diretrizes utilizadas para o desenvolvimento de modelos. Também inclui uma notação textual e uma notação gráfica para modelagem de sistemas orientados a objetos. A notação provê mecanismos para modelar herança e relações entre classes, e tem uma pequena coleção de técnicas para expressar relacionamentos dinâmicos.

Os primeiros passos no processo recomendado BON fazem uso da notação informal de cartões (*CRC index cards*) para documentar potencias classes, agrupamento de classes e propriedades das classes. Passos intermediários contam com as notações de modelagem estática e dinâmica. Finalmente, o último passo consiste no mapeamento do modelo BON para uma linguagem de programação orientada a objetos[23].

2.3.1 Cartões de Modelagem BON

Existem dois tipos de modelos na notação BON:

1. **Estático:** Descrição da estrutura do sistema: quais são os componentes e como eles se relacionam. Eles são independentes do tempo ou representam um *snapshot* em um certo momento.

2. **Dinâmico:** Exibem o comportamento do sistema no tempo. Em um contexto orientado a objetos, isto significa como os objetos interagem em tempo de execução.

Os cartões de modelagem são do tipo estático. Estes modelos são utilizados no início do processo para comunicar as idéias para pessoas sem conhecimento técnico como usuários finais, consumidores e especialistas no domínio. Além disso, os modelos também podem ser utilizados mais tarde como documentação em alto nível do sistema e serem guardados com descrições mais formais por uma ferramenta case. Existem três tipos de cartões de modelagem no modelo estático:

2.3.1.1 Cartão de sistema

O cartão de sistema (um por sistema) contém uma breve descrição de cada *cluster* do sistema em alto nível. Em BON um sistema é constituído por um ou mais grupos ou *clusters*, cada um contendo um número de classes ou *subclusters*.

2.3.1.2 Cartão de *cluster*

Um cartão de *cluster* contém uma breve descrição de cada classe e *subcluster* pertencentes ao *cluster*. Nomes de *subclusters* são colocados entre parênteses para distingui-los das classes.

2.3.1.3 Cartão de classe

Este é o elemento da notação BON utilizado neste trabalho para modelagem das classes a a partir das histórias de usuário. Um cartão de classe modela uma classe individualmente. As classes são vistas como caixas pretas e as informações presentes nesses cartões são as respostas das seguintes perguntas:

- Que informações outras classes podem solicitar desta classe? Estas respostas são traduzidas como consultas (*queries*) aplicáveis a esta classe.
- Que serviços outras classes podem solicitar desta classe? Estas respostas são traduzidas como comandos (*commands*) aplicáveis a esta classe.
- Quais regras devem ser obedecidas pela classe e seus dependentes? Estas respostas são traduzidas como restrições (*constraints*) da classe.

A Figura 4 mostra um exemplo de cartão para modelar uma classe citizen (cidadão), a partir do mapeamento, das respostas das perguntas descritas anteriormente, para as consultas, comandos e restrições da classe.

CLASS	*CITIZEN*	**Part:**1/1
TYPE OF OBJECT: Person born or living in a country	**INDEXING:** **cluster:** CIVIL_STATUS **created:** 1993-03-15 jmn **revised:** 1993-05-12 kw	
Queries	Name, Sex, Age, Single, Spouse, Children,Parents.	
Commands	Marry. Divorce.	
Constraints	Each citizen has two parents. At most one spouse allowed. May not marry children or parents or person of same sex. Spouse's spouse must be this person All children, if any, must have this person among their parents.	

Figura 4. Cartão de classe para modelagem de classe *Citizen* (adaptado de [23]).

2.4 XMI

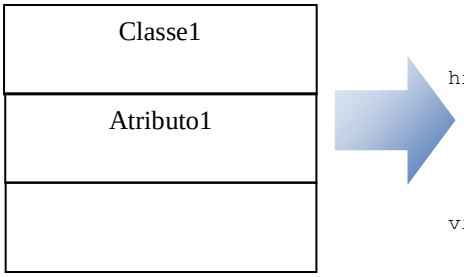
O *XML metadata interchange* (XMI)[21] é uma linguagem baseada em XML e seu principal objetivo é facilitar o intercâmbio de *metadados*, termo genérico para qualquer dado que de alguma forma descreve uma informação entre ferramentas de modelagem baseadas na UML e repositório de *metadados*, baseados no Meta-Object Facility (MOF)[25], em um ambiente distribuído e heterogêneo. O XMI integra três padrões da indústria de software:

- XML, um padrão da W3C, que é uma linguagem de marcação extensível.
- UML, um padrão da OMG, que define uma linguagem de modelagem orientada a objetos
- MOF, um padrão da OMG, que é uma estrutura de definição de modelos de *metadados* e fornece ferramentas para interfaces programáveis de armazenamento e acesso a *metadados* em repositórios.

A integração destes três modelos dentro do XMI permite o compartilhamento de objetos e outros *metadados*, visto que as ferramentas poderão transferir, compreender e utilizar modelos criados com outras ferramentas.

XMI é um padrão de comunicação entre ferramentas de modelagem, por isso, o protótipo desenvolvido neste trabalho, utiliza esta linguagem para descrever o diagrama de classes gerado a partir das histórias.

A Figura 5 mostra a modelagem de uma classe hipotética (Classe1) com a utilização de XMI.




```
<XMI xmi.version="1.1" xmlns:UML="org.omg/standards/UML">
   <XMI.header>
      <XMI.metamodel name="UML" version="1.3" href="UML.xml"/>
      <XMI.model name="example" version="1"
href="example.xml"/>
   </XMI.header>
   <XMI.content>
      <UML:Class name="Classe1">
         <UML:Classifier.feature>
            <UML:Attribute name="Atributo1"
visibility="private"/>
         </UML:Classifier.feature>
      </UML:Class>
   </XMI.content>
</XMI>
```

Figura 5. Exemplo de utilização de XMI na modelagem de uma classe

2.5 XSLT

A *Extensible Markup Language* XML[24] é uma linguagem de descrição de dados que se concentra na estrutura da informação, e não na sua aparência. Assim, para visualizarmos documentos XML precisamos formatá-los ou estilizá-los. *XML Stylesheet Language Transformation* (XSLT)[24] é a recomendação do *World Wide Web Consortium* (W3C) para este fim, pois aceita a transformação do documento antes da exibição. A XSL normalmente é utilizada para estilos avançados, como a criação de sumários em documentos, por exemplo. Na prática, esta linguagem especifica a transformação de documentos XML. Ela recebe um documento XML e o transforma em outro documento XML, conforme ilustrado na Figura 6.

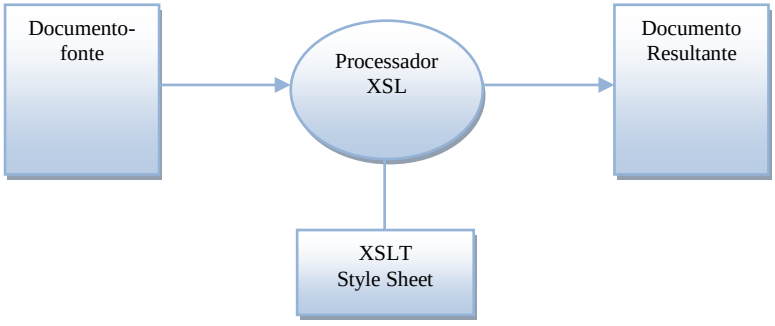


Figura 6. Utilização de XSL para transformar um documento XML

Como já foi dito, a XSLT não está limitada a atividades de estilo, visto que muitas aplicações exigem a transformação de documentos. Assim, XSLT pode ser utilizada também para:

- Incluir elementos especificamente para exibição, como o logotipo ou o endereço do emissor de uma fatura em XML.
- Criar conteúdo a partir de outro existente, como ao criar o sumário.
- Oferecer diferentes níveis de detalhes das informações de um documento XML com a utilização de várias folhas de estilos.

- Transformar documentos XML para HTML, para torná-los compatíveis com os navegadores existentes.

Utilizaremos XSLT para gerar o arquivo XMI que descreve o o diagrama de classes, a partir de outro documento XML gerado a partir das histórias cadastradas.

Capítulo 3

Uma Ferramenta para registro de histórias

Neste capítulo, descrevemos a implementação do protótipo desenvolvido neste trabalho de conclusão de curso, além de indicarmos o ponto de inserção da ferramenta no processo de desenvolvimento XP. Também serão abordadas as vantagens que podem ser obtidas por equipes de desenvolvimento ao adotarem a ferramenta em seu processo de construção de sistemas.

3.1 Visão Geral do Processo de Desenvolvimento

Como apresentado no Capítulo 2, o XP é um método iterativo e no início de cada iteração, o cliente explicita funcionalidades, ou histórias de usuário, a fim de que os desenvolvedores as implementem no decorrer desta mesma iteração.

Também foi apresentado que as histórias de usuário servem apenas como lembretes sobre as funcionalidades e que são estimadas pela equipe de desenvolvimento, além de serem criados testes de aceitação para elas antes mesmo da codificação.

A utilização da ferramenta proposta acontece nesta fase de planejamento da iteração. O registro das histórias é realizado e com ele a estimativa da equipe. Também podem ser armazenadas as descrições de testes para validação da implementação, contudo a ferramenta apenas registra essas descrições e cabe à equipe de desenvolvimento realizar os devidos testes.

Após o jogo do planejamento[5] e a conseqüente escolha das histórias priorizadas para a iteração corrente, dá-se início à codificação. Durante toda esta fase a equipe tem acesso direto a um especialista de negócio e é ao decorrer desta mesma fase que os detalhes das funcionalidades são entendidos por ela. Em outras palavras, é nesta fase que se intensifica a comunicação entre cliente e desenvolvedor a fim de detalharem o problema e então desenvolver a solução.

A ferramenta proposta permite este aprofundamento nas histórias através de seus detalhamentos. Cada detalhamento da história corresponde a uma classe, como será explicado

mais adiante. Com estes detalhamentos torna-se possível a geração de um arquivo XMI pela ferramenta proposta, que pode ser importado por uma ferramenta de modelagem, a exemplo do ArgoUml[9].

O modelo gerado pode ser utilizado para as discussões com o cliente sobre as funcionalidades em questão, uma vez que um diagrama de classes é descrito em uma linguagem gráfica que pode ser facilmente entendida por pessoas sem conhecimento técnico[1].

Esta visualização das entidades do sistema torna a comunicação entre cliente e desenvolvedor mais eficiente, visto que a discussão pode ser refletida em algo mais concreto, o que facilita o entendimento e análise.

Após a concordância parcial entre desenvolvedores e cliente sobre a definição inicial dos aspectos discutidos da funcionalidade, a equipe pode utilizar uma ferramenta MDA para gerar, no mínimo, as classes iniciais do sistema. Ressaltamos, porém, que algumas ferramentas MDA, como o BaseGen[8], geram não somente as classes, mas o banco de dados e interface gráfica[8].

Com a geração automática das classes do sistema, a equipe continua com a codificação relativa à parte de negócio e então realiza os testes, ao mesmo tempo em que pode consultar o cliente ou seu representante, e realizar modificações no diagrama de classes caso seja necessário.

Desta forma, as etapas subseqüentes ao detalhamento das histórias, devem ser as que já são praticadas por equipes de desenvolvimento que utilizam XP.

3.2 A Ferramenta

O protótipo foi desenvolvido na linguagem C#.net[26], sendo utilizado o *Visual Studio* como ambiente de desenvolvimento.

Entre as vantagens da linguagem escolhida podemos citar a disponibilização de uma diversidade de bibliotecas de funções úteis para este projeto, como as bibliotecas nativas de manipulação de XML e XSLT. Além disso, contamos com vantagens possíveis através de um desenvolvimento orientado a objetos, a exemplo de herança e interface.

O *Visual Studio* também trouxe benefícios ao desenvolvimento. Entre outros, utilizamos recursos como a edição gráfica de um esquema XML, utilizado no projeto para criar a estrutura de persistência (*DataSet*) como arquivos XML. Esta estrutura de persistência será apresentada mais adiante.

Além disso, com o *Visual Studio*, foi facilitada a modificação de controles nativos por nós através do mecanismo de herança visual. Esta modificação de controles foi realizada para modularizar o código de acesso aos dados armazenados. Ainda, utilizamos este ambiente para geração/edição do arquivo XSLT utilizado no processo de transcrição para o arquivo XMI.

A Figura 7 exibe a tela principal do protótipo e a tela de cadastro de projetos. Esta última, em particular, é a tela que permite a geração do arquivo XMI. Contudo, para que essa geração seja possível, devem ser cadastrados o projeto, as histórias associadas a ele e os detalhes de cada uma destas histórias. Ressaltamos que a geração do arquivo XMI é realizada por projeto.

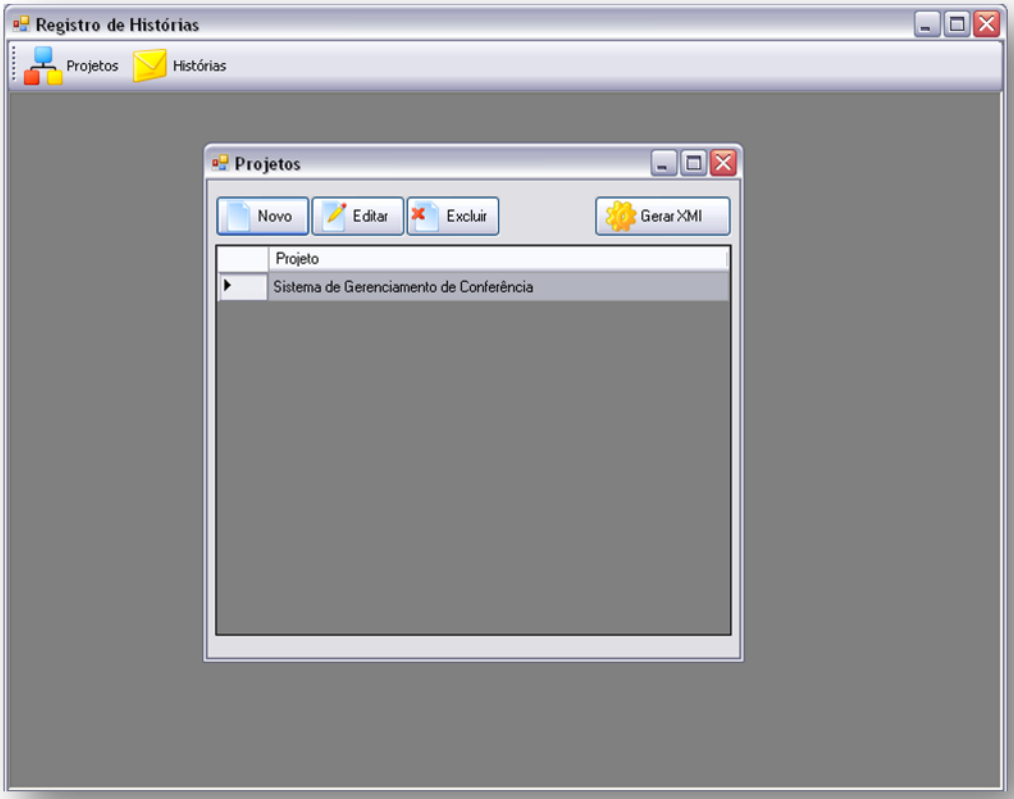


Figura 7. Tela principal e tela de cadastro de projetos

Após o cadastramento do projeto, podemos inserir as histórias sugeridas pelo cliente. Algumas destas histórias cadastradas serão escolhidas para serem implementadas na iteração atual, mas todas elas devem ser estimadas pela equipe de desenvolvimento independentemente da precisão desta estimativa. A tela de cadastro de histórias é exibida na Figura 8. As seguintes informações são armazenadas para cada histórias:

- **Projeto:** Informa o sistema ao qual esta história está relacionada.
- **História Pai:** Caso as histórias sugeridas pelo cliente sejam complexas a ponto de dificultar a estimativa, as histórias podem ser decompostas em outras menores. Este campo indica se uma história é decomposição de outra.
- **Data de Criação:** Data em que o cliente criou a história.
- **Quantidade Dias:** Estimativa, em dias, para implementação da história.
- **Status:** Indica se a história já começou a ser implementada ou se já foi finalizada.
- **Descrição:** Uma breve descrição informal sobre a funcionalidade requerida pelo cliente. Podem também ser especificados testes a fim de validar a implementação.

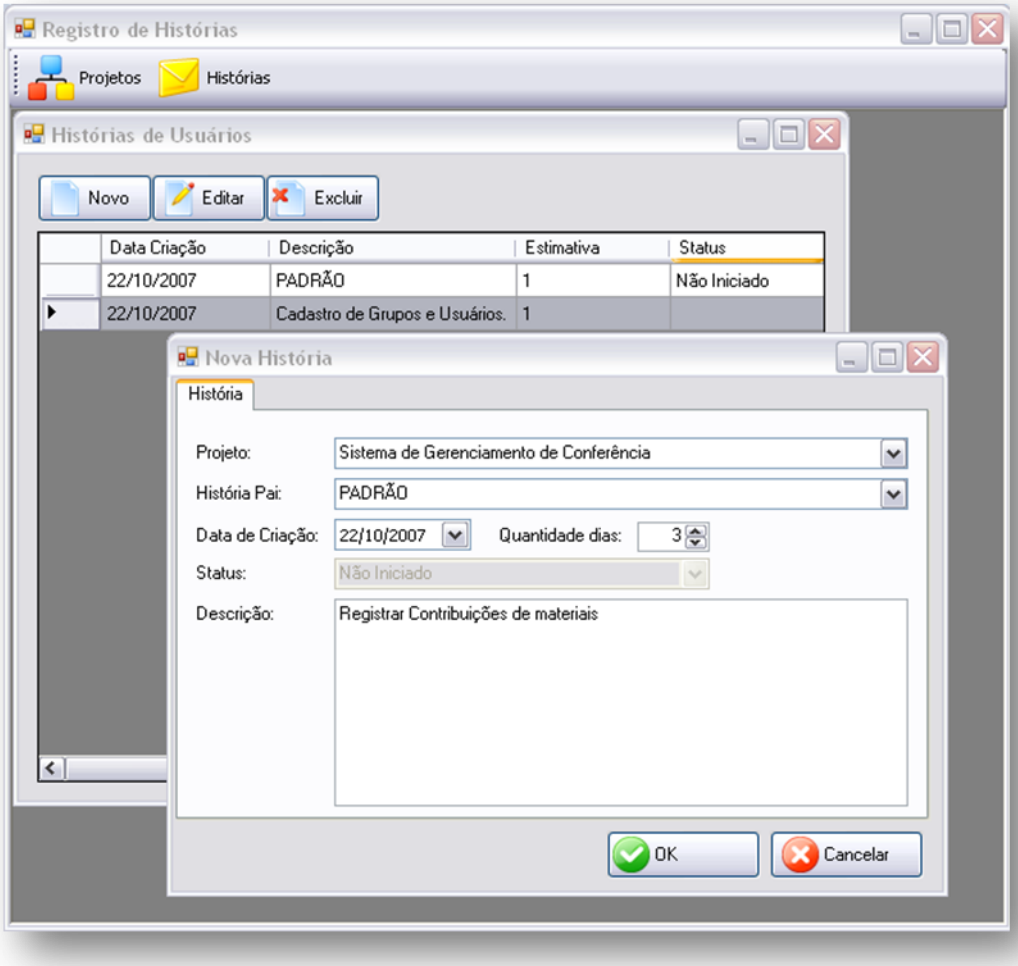


Figura 8. Tela de Registro de Histórias

O detalhamento das histórias é o próximo passo. Uma história pode possuir vários detalhes e cada um destes será mapeado para uma classe do sistema. As informações mantidas pelo registro desses detalhes são inspiradas na notação BON apresentada no Capítulo 2. A Figura 9 exibe a tela de cadastro dos detalhes de uma história e a seguir será exibido como estes detalhes foram modelados:

- **História:** Indica a história com a qual esse detalhe se relaciona
- **Descrição:** Contém uma tag com o nome da classe (<NomeClasse>) e uma descrição informal deste detalhe. A ordem não é importante e a tag pode estar incluída no texto de descrição.
- **Consultas:** Indica as informações que a classe poderá oferecer. Estas consultas serão mapedas em atributos da classe e os seus tipos não são descritos.
- **Comandos:** Indica os serviços que esta classe provê para as outras. Estes comandos são mapeados em métodos da classe.

- **Restrições[1]:** Indica as regras que devem ser obedecidas pela classe e seus dependentes. Para uma classe "Grupo", por exemplo, poderíamos estabelecer que só fossem aceitas pessoas com idade superior a 21 anos.

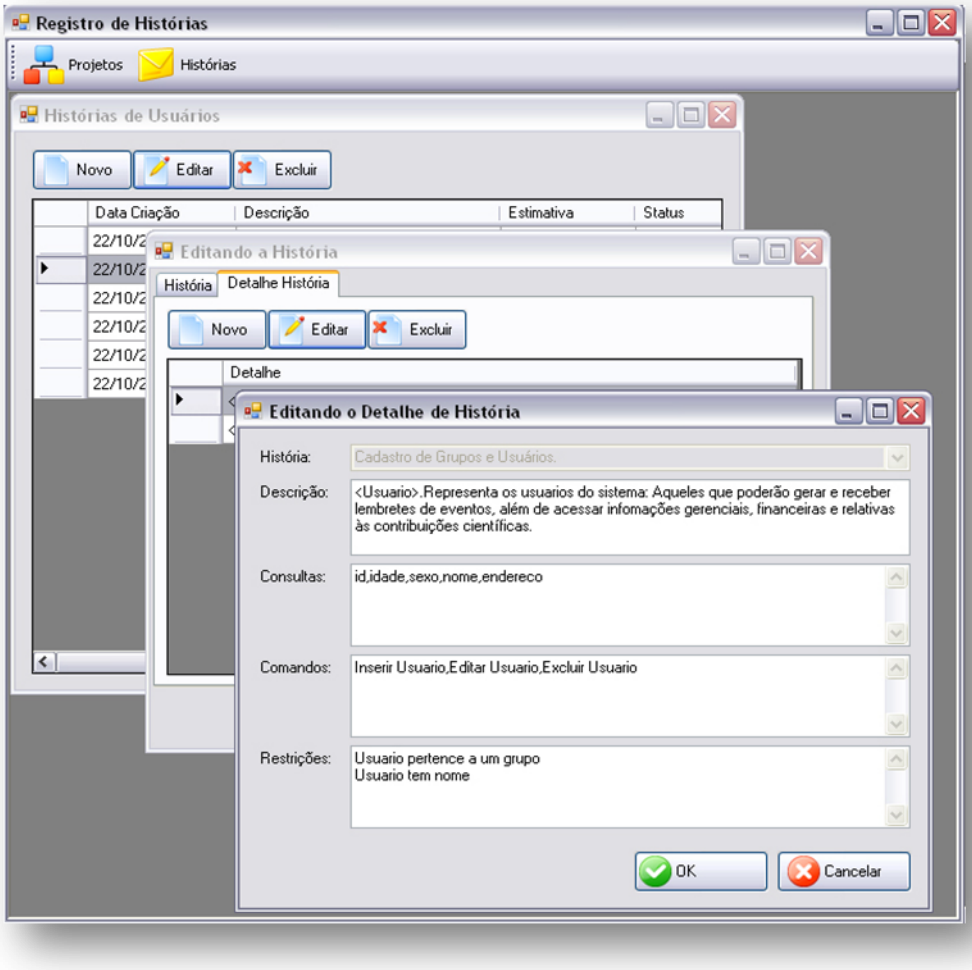


Figura 9. Tela de registro de detalhe da história

Cadastradas as histórias e seus detalhes, o desenvolvedor pode então gerar o modelo abstrato do sistema na tela de Projetos. Esta geração será exemplificada no próximo capítulo através de uma prova de conceito. A seguir, detalharemos os aspectos técnicos das principais funcionalidades do protótipo, a saber, a alimentação de dados e a transformação destes para a geração do XMI.

3.2.1 Alimentação de Dados

Foi apresentado como utilizar o sistema para cadastrar as informações relevantes. Nesta subseção detalharemos a modelagem desses dados e como eles são persistidos.

[1] As restrições ainda não são utilizadas na geração diagrama de classes. Apenas é possível armazená-las.

A estrutura de persistência foi criada com a utilização do *Visual Studio* e pode ser observada na Figura 10. Foi criado um esquema XML para descrever o modelo relacional entre os dados, através do componente *DataSet* disponível neste ambiente de desenvolvimento. Este componente permite: (i)a criação de estruturas que modelam as tabelas, chamadas de *DataTables*, e (ii)a definição de relacionamentos entre estas, como em um modelo relacional de banco de dados.

Além disso, este componente também provê serviços nativos de escrita e de leitura no formato XML, o que nos permite armazenar os dados em arquivos neste formato. As informações que persistem no protótipo e suas respectivas tabelas são descritas a seguir:

- **Projetos**, armazenados na tabela *dtProject*.
- **Histórias**, armazenadas na tabela *dtHistory*.
- **Detalhes das Histórias**, armazenadas na tabela *dtHistoryDetail*.

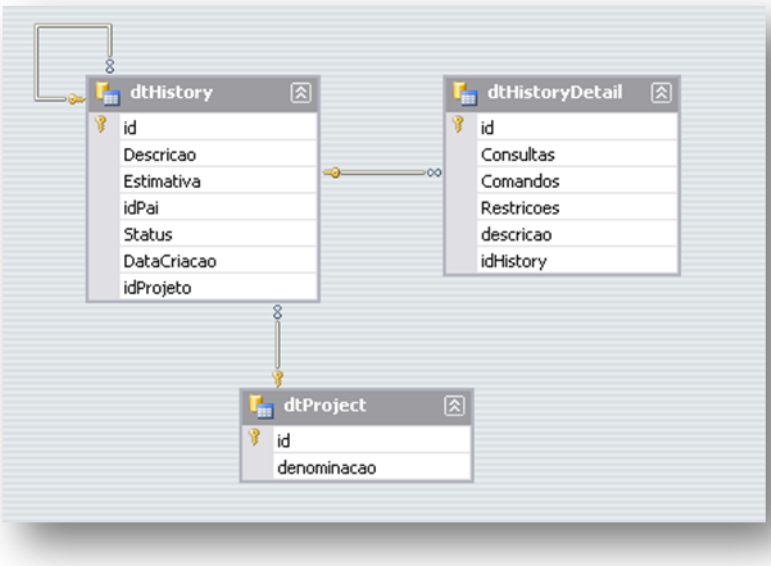


Figura 10. Estrutura de persistência gerada no *Visual Studio*

Como observado na Figura 10, um projeto pode conter várias histórias. Estas, por sua vez, podem se relacionar com outras. Isto é particularmente importante para garantir a qualidade das histórias escritas, discutida em detalhes no segundo capítulo. Este relacionamento serve apenas para que histórias possam ser decomposta em outras menores, mas facilmente estimáveis, por exemplo.

Os detalhes são utilizados para modelar as classes e, como uma história pode requerer mais de uma classe, o relacionamento entre história e detalhes é de um para muitos.

3.2.2 Transformação dos Dados

Detalharemos nesta subseção, como foi projetada a transformação dos dados, a fim de gerar o arquivo XMI do modelo abstrato.

Foi necessário mapear os dados armazenados para um documento XML a fim de facilitar a transformação. O esquema desta estrutura intermediária pode ser visualizado na Figura 11. O campo "Projeto" é o mais externo no XML e é decomposto em classes e associações entre elas. A estrutura das classes por sua vez é refletida nos conteúdos dos campos "Nome", "Atributo" e "Método".

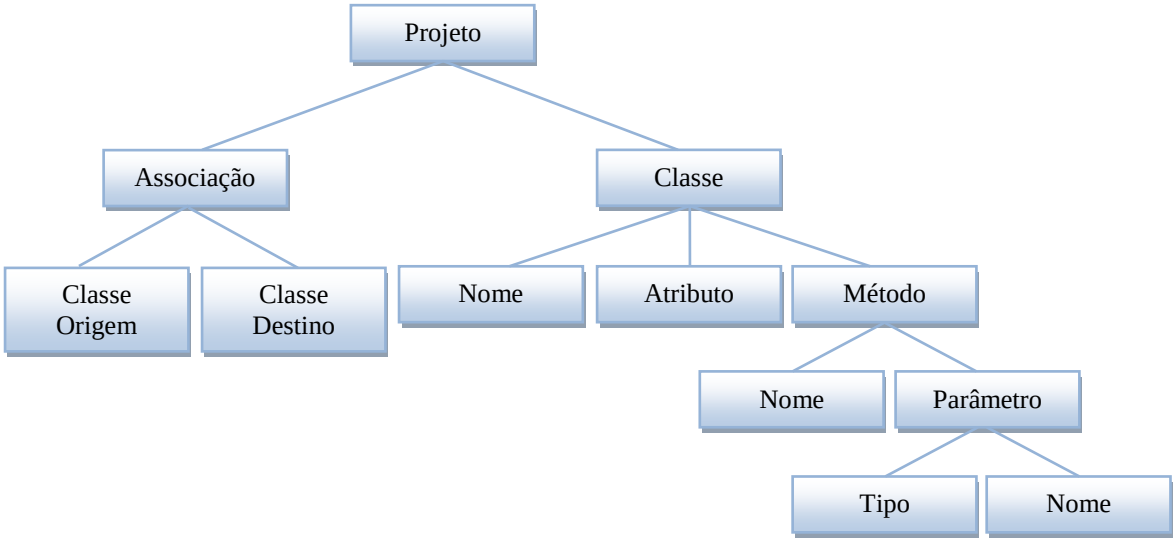


Figura 11. Árvore da estrutura intermediária para transformação

Além da construção dessa estrutura intermediária a partir dos detalhes das histórias, também foi necessário definir uma folha de estilo XSLT para mapear esta estrutura no arquivo XMI final. O arquivo *xslGenerateXMI.xslt* foi criado para isso e apesar de não definir possíveis transformações para todos os elementos de um diagrama de classes, permite o mapeamento dos atributos e métodos das classes e das associações entre elas.

A transformação acontece em dois passos. Primeiramente, o sistema percorre os detalhes das histórias associados ao projeto em questão e monta a estrutura intermediária na memória. Em seguida, utilizamos o processador XSLT definido no *namespace System.Xml.Xsl* de .net, passando para ele a estrutura intermediária e a folha de estilo XSLT. O resultado da transformação XSLT é salvo no diretório do executável e já pode ser importado por ferramentas de modelagem.

A construção da estrutura intermediária, no entanto, foi projetada sobre alguns pré-requisitos descritos a seguir:

- As consultas e comandos das classes devem estar separados por vírgula ou quebra de linha.
- Cada comando é descrito em língua natural. O nome do método será o nome do comando, substituindo os espaços em branco pelo caractere "_". O parâmetro do

método só é mapeado se o nome do comando contiver o nome da própria classe ou de outra no mesmo projeto. O tipo de retorno sempre é *void*.

- A descrição de cada detalhe deve conter, em qualquer posição, uma tag informando o nome da classe. Para uma classe de nome “Animal”, por exemplo, a tag seria <Animal> e esta poderia ser utilizada normalmente no texto de descrição da funcionalidade.
- Para associar classes, o campo de consulta da classe que referencia deve conter o nome de outra classe definida no protótipo para o mesmo projeto.

3.2.3 Limitações do Protótipo

O protótipo desenvolvido objetiva implementar a geração automática de um diagrama de classes com sua estrutura básica. Contudo, existem muitos aspectos importantes que precisariam ser incorporados à ferramenta para obter um maior benefício com a sua utilização. Entre as limitações deste protótipo, destacamos:

- O cadastro de detalhes de histórias é dificultado pelo fato de não se prover uma maneira de validá-lo a fim de permitir a geração correta da estrutura intermediária.
- Não se tem um controle das iterações e do desenvolvimento das histórias ao longo das mesmas. Assim, não conseguimos relacionar histórias e iterações, o que seria uma informação útil ao longo do processo além de permitir alterações em iterações específicas.
- A ferramenta sempre gera o diagrama de classes completo e sobrescreve o XMI gerado anteriormente. Assim, caso o diagrama tenha sido aperfeiçoado com o auxílio de uma ferramenta de modelagem, as alterações serão perdidas.
- Ainda não é possível descrever as restrições da classe, o que seria realizado com língua natural controlada. Estas restrições seriam então mapeadas para o modelo abstrato.

Capítulo 4

Prova de Conceito

Este capítulo aborda a utilização do protótipo desenvolvido neste trabalho para a geração automática de um diagrama de classes a partir das funcionalidades iniciais de um sistema de gerenciamento de conferência. O modelo gerado será importado por uma ferramenta MDA e as classes iniciais serão geradas.

4.1 Sistema de Gerenciamento de Conferência

Este sistema visa controlar aspectos essenciais para gerenciamento e controle de uma conferência científica internacional. O sistema foi adaptado de um dos estudos de caso para utilização da notação BON na modelagem de sistemas orientados a objetos[23] e deve ajudar os organizadores de uma conferência a lembrar de uma série de eventos durante a preparação do programa técnico, além de gerenciar contribuições científicas e inscrições. Os objetivos gerais do sistema são descritos a seguir:

- Monitorar os eventos agendados e garantir que todas as ações necessárias são realizadas a tempo.
- Automatizar o processo da maioria das tarefas da conferência evitando duplicidade de esforço e produzindo avisos a fim de garantir o cumprimento dos prazos.
- Servir como repositório de informações úteis para ambos, comitês técnico e organizacional.

Estas tarefas relativas à conferência podem ser divididas entre três grupos razoavelmente independentes:

- **Comitê de Programa**: Grupo responsável pelos assuntos técnicos. Entre outros devem convidar pesquisadores e desenvolvedores interessados em contribuir com material científico. Além disso, devem encaminhar os artigos para revisores, selecionando contribuições para incluir no programa final e agrupá-las em seções.

- **Comitê de Organização**: Grupo responsável por políticas gerais e logísticas. Isto inclui propagandas e e-mails em geral, decisões de preço e capacidade, patrocínio, lanches e almoços, reservas de acomodações e cartas de confirmação.
- **Departamento de Contabilidade:** Grupo responsável por questões financeiras. Isto inclui faturamento, realizar pagamentos e analisar estatísticas de vendas e gastos.

A Tabela 1 relaciona as principais informações de entrada e saída do sistema.

Tabela 1. Informações de entrada e Saída do sistema de Gerenciamento de Conferência

ENTRADAS	SAÍDAS
• Registro de participantes realizado através da internet. • Realização de pagamento através de cartão de crédito ou boleto. • Cadastro de contribuinte e submissão de seus artigos científicos. • Relatório de revisor de artigos.	• Chamada de artigos, convites, material promocional. • Cartas de confirmação, aceitação ou rejeição. • Tickets para inscritos. • Programa final. • Faturamento, lembretes. • Lista de inscritos, situação financeira.

4.2 Desenvolvimento

Abordaremos nesta seção como foi utilizada a ferramenta proposta neste trabalho, para o desenvolvimento de algumas funcionalidades iniciais do sistema de gerenciamento de conferência citado.

4.2.1 Histórias do Sistema

Baseados nas descrições do sistema, criamos algumas histórias de usuário a fim de simular a primeira iteração do XP para a construção deste sistema. Entre as histórias mais essenciais destacamos:

- Gerar lembretes de eventos
- Registrar inscrições
- Listar inscritos
- Registrar contribuições de artigos
- Listar situação das contribuições
- Listar situação de inscrições, informando necessidades como a de reserva de hotel e proibições alimentares
- Cadastro de Despesas
- Faturamento

A história escolhida para testarmos o protótipo foi a de gerar lembretes de eventos. Contudo, a fim de melhorarmos a precisão das estimativas, subdividimos a mesma em outras duas:

- Cadastrar Usuários e Grupos do sistema
- Permitir cadastro de lembrete para usuário ou Grupo

O registro da história principal foi realizado com a utilização da ferramenta. Em seguida cadastramos as duas histórias citadas, como sub-histórias desta principal. Como apresentado anteriormente, isto é possível através do campo "História Pai" na tela de cadastro de histórias.

Em seguida, cadastramos os detalhes da primeira sub-história, modelando as classes relacionadas à implementação desta funcionalidade. Na Figura 12 encontram-se as informações cadastradas para a classe "Usuario". No campo descrição encontra-se uma tag com o nome da classe, seguida de uma descrição do que ela representa no mundo real. As consultas e comandos também são descritos e encontram-se separados por vírgula.

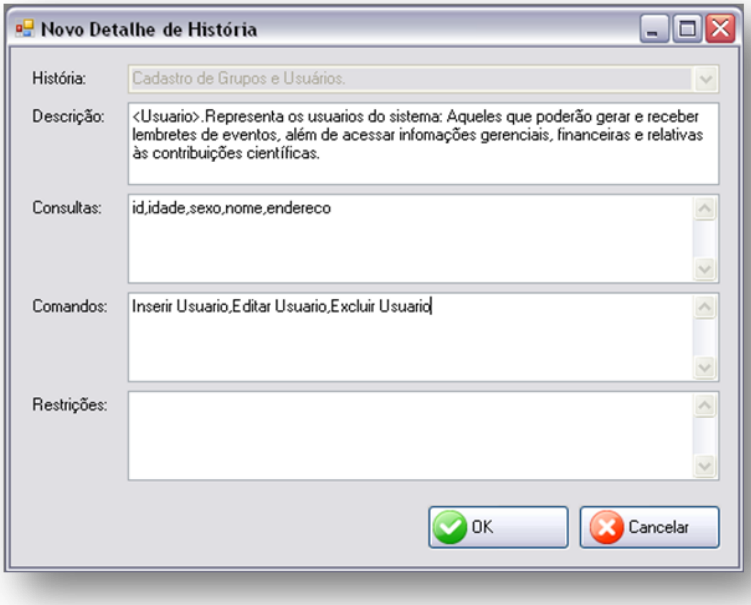


Figura 12. Especificação da classe "Usuario"

No momento da geração do arquivo XMI, o protótipo avalia se existem consultas descritas para uma classe, que contenham o nome de outra classe pertencente ao mesmo projeto. Caso existam, estas consultas serão mapeadas para associações entre as devidas classes, ao invés de um simples atributo. As associações sempre têm cardinalidade 1 para n, sendo n o número de objetos, na associação, para a classe referenciada.

Um exemplo disso pode ser observado na classe "Grupo"que também foi modelada para implementar a história escolhida, tendo seu detalhamento exibido na Figura 13. A consulta

"usuarios" é entendida como uma associação entre a classe "Grupo" e a classe "Usuario", sendo a cardinalidade 1 para n, ou seja, um objeto do tipo "Grupo" se relaciona com zero ou vários objetos do tipo "Usuario". É importante observar ainda que os comandos estão separados por quebra de linha, o que também é suportado pelo gerador do XMI.

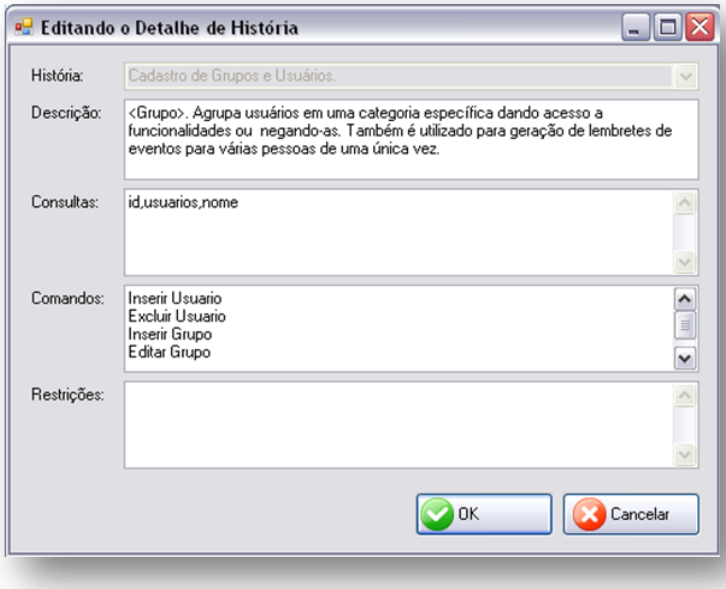


Figura 13. Especificação da classe "Grupo"

A segunda sub-história foi detalhada conforme Figura 14. A classe "Lembrete" referencia as classes "Usuario" e "Grupo".

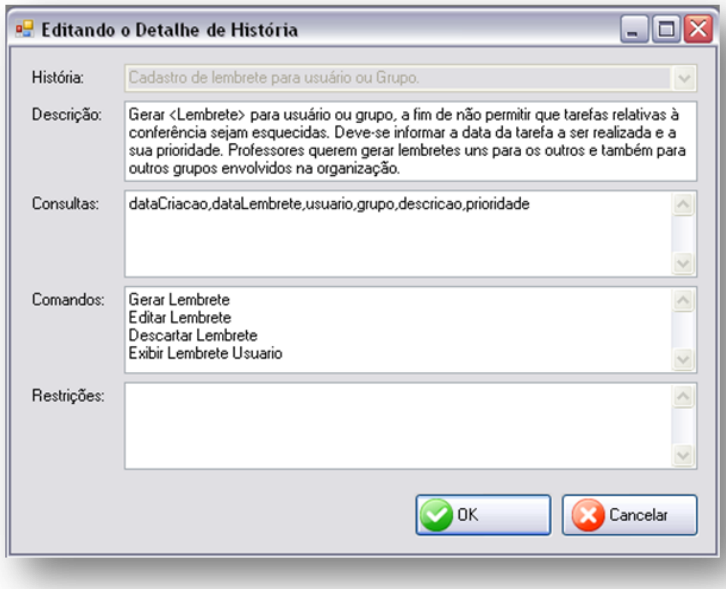


Figura 14. Especificação da classe "Lembrete"

4.2.2 Importação do Modelo

Cadastradas as histórias desejadas e seus respectivos detalhes, utilizamos a funcionalidade de geração de arquivo XMI disponibilizada na tela de Projetos. O arquivo "Sistema de Gerenciamento de Conferência.xmi" foi então gerado no diretório do protótipo.

Em seguida, utilizamos uma ferramenta MDA para importar o XMI gerado. O diagrama de classes exibido na Figura 15 mostra o resultado desta importação realizada com o ArgoUML. Os nomes de atributos e métodos foram gerados corretamente, assim como a relação entre as classes. Contudo, podemos também observar as limitações do protótipo, no que diz respeito, entre outros, aos tipos de retorno dos métodos e aos tipos dos atributos que precisam ser modificados no próprio modelo.

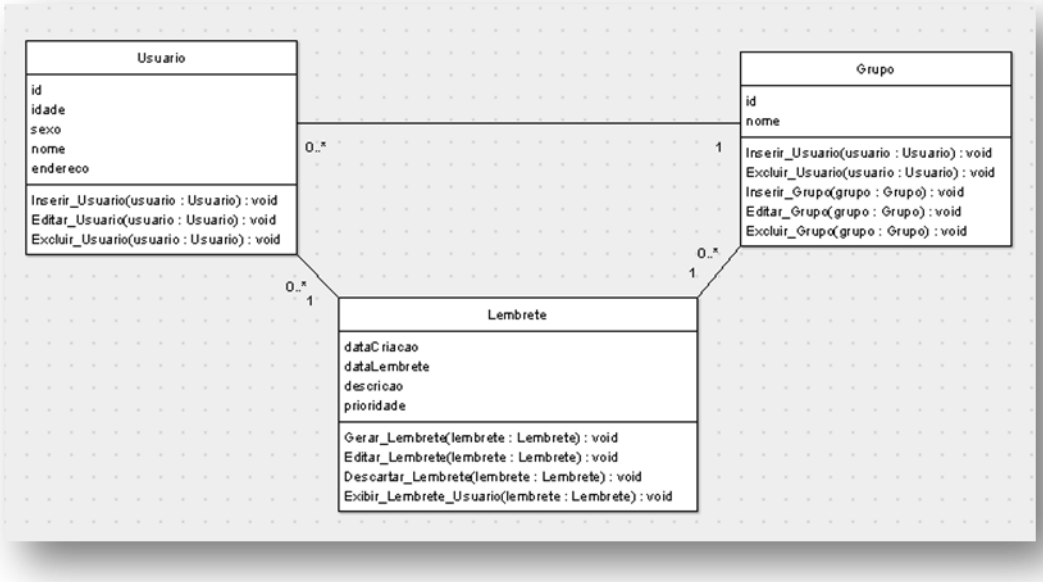
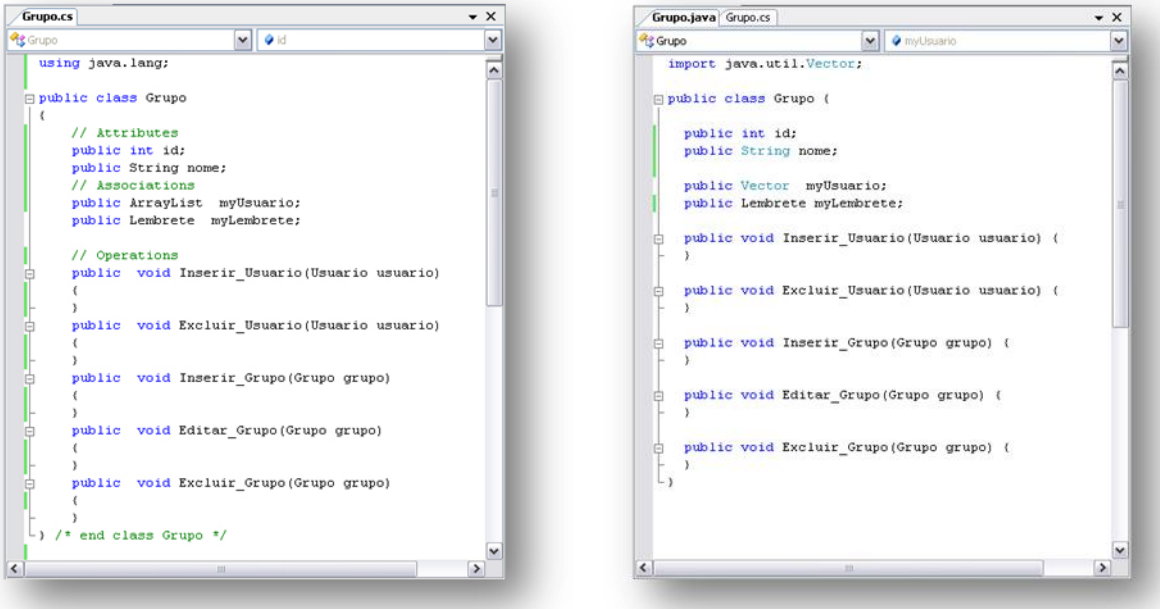


Figura 15. Diagrama de classes gerado com ArgoUML a partir do XMI gerado

4.2.3 Geração das Classes

Após informarmos os tipos corretos dos atributos no ArgoUML, geramos as classes que implementam este modelo abstrato gerado pela nossa ferramenta. A Figura 16 mostra o resultado da geração de código realizada pelo ArgoUML sobre o modelo, para a classe "Grupo". A geração foi feita em duas linguagens: Java e C#.

Observamos que a geração da classe na linguagem Java ficou mais correta, uma vez que em C#, a classe gerada importa uma biblioteca existente apenas na linguagem Java. Este é um problema da ferramenta ArgoUML e provavelmente não ocorreria se outra ferramenta de modelagem tivesse sido empregada.



(a) (b)

Figura 16. (a) Classe “Grupo” na linguagem C# , (b) Mesma classe em Java

Ressaltamos que a geração de código pode ser realizada por qualquer ferramenta MDA capaz de entender XMI, que é o padrão utilizado para comunicação entre ferramentas de modelagem.

Desta maneira, outras ferramentas como o BaseGen mostram-se interessantes para este propósito de geração semi-automática de código, uma vez que sua utilização nos proporcionaria não apenas as estruturas das classes, mas também código de acesso a banco de dados e interface gráfica. A geração semi-automática de código também se mostra interessante para manter padrões de codificação o que é uma das recomendações do XP.

Capítulo 5

Conclusões e Trabalhos Futuros

Este capítulo sumariza as contribuições deste trabalho, destacando as vantagens e limitações da ferramenta de registro de histórias proposta, comentando os resultados obtidos com sua utilização na prova de conceito e indicando trabalhos futuros.

5.1 Conclusões

Foi apresentado que o XP é um método ágil amplamente utilizado na condução de projetos de software e que sua utilização, entre outros, torna o processo mais adaptável a mudanças no escopo ao mesmo tempo em que diminui os riscos do produto não satisfazer as necessidades do cliente. Ainda, o desenvolvimento XP é dividido em iterações e no início de cada iteração o cliente explicita funcionalidades, ou histórias de usuário, a fim de que a equipe de desenvolvimento as estime. Algumas histórias são escolhidas para serem implementadas na iteração atual.

Este trabalho objetivou a criação de um protótipo de ferramenta de registro de histórias XP, permitindo o gerenciamento das mesmas. Esta ferramenta também implementa a geração automática de um diagrama de classes a partir destas histórias previamente cadastradas. Foram apresentados detalhes técnicos sobre o projeto da ferramenta e sua implementação, ao longo do trabalho. Além disso, discutimos, no Capítulo 3, as tecnologias utilizadas na sua construção.

Discutimos o processo de desenvolvimento com a utilização da ferramenta, ou seja, analisamos o ponto de inserção da ferramenta no processo XP. A importância de sua utilização no planejamento e execução das iterações foi então abordada.

Uma prova de conceito foi realizada com a utilização da ferramenta para um Sistema de Gerenciamento de Conferência. Após o cadastro das histórias e seus detalhes, geramos o modelo abstrato e o importamos com uma ferramenta de modelagem, a partir da qual, geramos código em C# e Java. Desta forma, mostramos que foi possível mapearmos atributos, métodos e associações

entre classes, apenas com o cadastro das histórias e seus detalhes em linguagem natural, com a utilização de alguns artifícios como tags e separadores.

Também citamos a possibilidade de importação do modelo gerado pelo protótipo, através de ferramentas MDA, a exemplo do BaseGen, e discutimos as vantagens de sua utilização.

5.2 Trabalhos Futuros

O protótipo desenvolvido apresenta algumas limitações, como discutido no capítulo 3. Contudo, este protótipo pode ser expandido em várias direções. A seguir são sugeridos alguns trabalhos futuros:

1. Utilizar língua natural controlada para especificar as restrições das classes, utilizando OCL para incluí-las no modelo gerado. Isto seria um benefício extremamente importante obtido com a adoção da ferramenta.
2. Também utilizando língua natural controlada, permitir a criação de classes de teste a partir da descrição das histórias.
3. Criar interface gráfica que possibilite a manipulação das histórias como cartões dispostos em uma mesa. Esta é a forma habitual de manipular histórias em XP.
4. Dar suporte a outros elementos de UML na criação do modelo. Apenas classes, atributos, métodos e associações são gerados atualmente.

Bibliografia

[1] SOMMERVILLE, Ian. Software Engineering. 6.ed. Lancaster: Addison-Wesley, 2001. 720 p.

[2] PRESSMAN, Roger S. Software engineering: a practitioner's approach. 4.ed. New York: McGraw-Hill, 1997. 885 p.

[3] JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. The Unified software development process: the complete guide to the Unified Process from the original designers. 1.ed. Reading, MA: Addison-Wesley, 1999. 463 p.

[4] MANIFESTO for Agile Software Development. Disponível em <http://www.agilemanifesto.org>. Acesso em 10.11.2007.

[5] BECK, Kent. Extreme Programming Explained: Embrace change. Reading, Massachusetts: 1.ed. Addison-Wesley, 2000. 190 p.

[6] BOOCH, G.; RUMBAUGH J.; JACOBSON, Ivar. The Unified Modeling Language User Guide: Addison-Wesley. 2005. 475 p. (Object Technology Series)

[7] OBJECT MANAGEMENT GROUP. Model Driven Architecture. 2007. Disponível em: <http://www.omg.br/mda>. Acesso em 10.09.2007.

[8] SILVA, RODRIGO. BaseGen: Uma ferramenta baseada em MDA para construção Semi-Automática de aplicações WEB. 2006. 178 p. Dissertação (Mestrado) – Universidade Federal da Paraíba, João Pessoa.

[9] COLLABNET. ArgoUML. Disponível em <http://argouml.tigris.org/>. Acesso em 01.11.2007.

[10] SCHWABER, KEN. SCRUM Development Process – Advanced Development Methods, 1997. Disponível em: <http://jeffsutherland.com/oopsla/schwapub.pdf>. Acesso em 01.09.2007.

[11] CRYSTAL. Crystal Methodologies. Disponível em: <http://crystalmethodologies.org> Acesso em 15.11.2007.

[12] ELSSAMADESY, AMR. Patterns of Agile Practices Adoption – The Technical Cluster, 2007.

[13] BECK, Kent; FOWLER, Martin. Planning Extreme Programming. 1.ed. Boston: Addison-Wesley, 2001. 139 p.

[14] POPPENDIECK, Mary; POPPENDIECK, Tom. Lean software development: an agile toolkit. 1.ed. Upper Saddle River, NJ: Addison-Wesley, 2003. 240 p.

[15] BECK, Kent; ANDRES, Cynthia. Extreme Programming Explained: Embrace Change. 2.ed. Addison Wesley Professional, 2004. 190 p.

[16] FOWLER, Martin. The New Methodology. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: 01.11.2007.

[17] COHN, MIKE. User Histories Applied. 1.ed. Addison-Wesley, 2004, p. 17-28.

[18] WAKE, William C. Extreme Programming Explored. Reading, Massachusetts: Ed. Addison-Wesley, 2002.

[19] OBJECT MANAGEMENT GROUP. The Object Management Group. Disponível em <http://www.omg.org>. Acesso em 03.11.2007.

[20] RAMLJAKM D., *et al*. Building Enterprise Information System Using Model Driven Architecture on J2ee Platform. Telecommunications, 2003. ConTel 2003. Proceeding of the 7th International Conference em 13.11.2003.521—526vol.2 p.

[21] OBJECT MANAGEMENT GROUP. Xml Metadata Interchange. 2006. Disponível em <http://www.omg.org/docs/formal/05-09-01.pdf>>. Acesso em 03.11.2007.

[22] WARMER, Jos; KLEPPE, Anneke. The Object Constraint Language: Getting Your Models Ready for MDA. Second Edition. Massachusetts: Ed.Addison Wesley, 2003. 240 p.

[23] WALDÉN, Kim e NERSON, J. Seamless Object-Oriented Software Architecture – Analysis and design of reliable systems. 1.ed. Prentice Hall, 1995. p. 3-30. (The Object-Oriented Series).

[24] MARCHAL, Benoît. XML – Conceitos e Aplicações. Ed.Berkeley, 2000.

[25] OBJECT MANAGMENT GROUP. OMG's MetaObject Facility. Disponível em <http://www.omg.org/mof>. Acesso em 10.11.2007.

[26] MICROSOFT. Microsoft Developer Network. Disponível em <http://msdn2.microsoft.com/pt-br/default.aspx >. Acesso em 02.11.2007.