UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DA PRODUÇÃO

METIMAGE: UMA METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE PARA O PROCESSAMENTO E ANÁLISE DE IMAGENS

DISSERTAÇÃO DE MESTRADO

Adriane Pedroso Dias Ferreira

Santa Maria, RS, Brasil
2006

METIMAGE: UMA METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE PARA O PROCESSAMENTO E ANÁLISE DE IMAGENS

por

Adriane Pedroso Dias Ferreira

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-graduação em Engenharia da Produção, Área de Concentração em Tecnologia da Informação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Engenharia da Produção.**

Orientador: Prof. PhD Marcos Cordeiro d'Ornellas

Santa Maria, RS, Brasil.

2006

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia da Produção

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

METIMAGE: UMA METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE PARA PROCESSAMENTO E ANÁLISE DE IMAGENS

elaborada por
Adriane Pedroso Dias Ferreira

como requisito parcial para obtenção do grau de
Mestre em Engenharia da Produção

COMISÃO EXAMINADORA:

Marcos Cordeiro d'Ornellas, PhD
(Presidente/Orientador)

Cesar Tadeu Pozzer, Dr. (UFSM)

Andréa Schwertner Charão, Dra. (UFSM)

Santa Maria, 24 de Novembro de 2006.

Dedico essa dissertação aos meus pais
Francisco Dias e Jane Dias,
aos meus avós (in *memorian*)
e ao meu irmão Francisco.

AGRADECIMENTOS

Primeiramente, tenho que dizer muito obrigada aos meus pais por sempre confiarem, acreditarem e me darem suporte emocional necessário para vencer os obstáculos. Vocês são muito importantes para mim.

Ao meu amor, Luciano Ferreira, por todo apoio, paciência, carinho e por escutar as minhas angústias nos momentos difíceis. Te Amo Muito!

Em especial, ao meu orientador, Prof. Dr. Marcos Cordeiro d'Ornellas, pelo incentivo, pela confiança em mim depositada, pelas oportunidades, pelo exemplo de profissionalismo, pelos conhecimentos transmitidos no transcorrer do curso e, acima de tudo, pela sua amizade.

Não podia deixar de lembrar das minhas amigas Sabrina e Patrícia, vocês estavam presentes nos momentos mais importantes dessa etapa da minha vida. Muito obrigada por todas as vezes que vocês me ouviram, me aconselharam... Obrigada pelas conversas, pela parceria para o chimarrão, enfim, vocês estão no meu coração.

Um agradecimento especial a Profª Cristiane que sempre nos dizia: "Calma gurias, vai dar certo...", obrigada pela força.

A todos os amigos do Grupo PIGS pelas dicas, conversas e dedicação.

Ao CNPq pelo apoio financeiro.

Descobri como é bom chegar quando se tem paciência,
e para chegar onde quer que seja,
aprendi que não é preciso dominar a força, mas a razão.
É preciso antes de mais nada, querer.
(Amyr Klink)

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Engenharia da Produção
Universidade Federal de Santa Maria

METIMAGE: UMA METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE PARA O PROCESSAMENTO E ANÁLISE DE IMAGENS

AUTORA: Adriane Pedroso Dias Ferreira
ORIENTADOR: Marcos Cordeiro d'Ornellas
Data e Local da Defesa: Santa Maria, 24 de novembro de 2006.

O desenvolvimento de software para processamento e análise de imagens é uma tarefa complexa por utilizar métodos matemáticos para resolver os problemas, por necessitar de uma equipe multidisciplinar e por exigir alto grau de qualidade do software desenvolvido. Portanto, fazer uso de uma metodologia que organize e melhore o processo de desenvolvimento desse tipo de software é de vital importância. A existência de uma metodologia é apontada como um dos primeiros passos em direção ao gerenciamento e a melhoria do processo de desenvolvimento de software. Assim, este trabalho apresenta uma metodologia específica para o desenvolvimento de software para processamento e análise de imagens, chamada nesse trabalho de MetImage. O objetivo dessa metodologia é suprir as deficiências detectadas nas metodologias existentes, tais como o excesso de recursos, burocracia, controle exagerado e falta de documentação, em alguns casos específicos. A metodologia proposta foi implantada no contexto de um grupo de pesquisa. Os principais resultados obtidos foram: a especificação das atividades da equipe, a inclusão de uma etapa de aprendizagem sobre os métodos matemáticos necessários para a implementação das funcionalidades requeridas pelos sistemas e a padronização de código. Além disso, a documentação gerada pode servir de apoio para o entendimento entre especialistas das diferentes áreas que fazem parte do grupo de pesquisa.

Palavras-chave: Engenharia de software, processamento e análise de imagens.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Production Engineering
Federal University of Santa Maria

METIMAGE: A METHODOLOGY FOR DEVELOPMENT OF IMAGE PROCESSING AND ANALYSIS SOFTWARE

AUTHOR: Adriane Pedroso Dias Ferreira
ADVISOR: Marcos Cordeiro d'Ornellas
Date and city: Santa Maria, 24th November 2006.

The development of image processing and analysis software is a complex task by using mathematical methods to solve problems, by needing multidisciplinary team and demanding high degree of software developed quality. Therefore, is very important to utilize a methodology that organizes and improves the process of development of this type of software. The existence of a methodology is pointed out as one of the first steps toward the management and improvement the process software development. Therefore, this work presents a specific methodology for the development of image processing and analysis software, called, in this work, MetImage. The goal of this methodology is to improve the deficiencies detected in existing methodologies, such as the excessive resources, bureaucracy, exaggerated control and the documentation gap, in same specific cases. The methodology proposal was implanted in the context of a research group. The main results obtained were the specification of the team activities, the inclusion of the stage of learning on the necessary mathematical methods for the implementation of the functionalities and the standardization of code. Moreover, the documentation generated can be use as a support for the agreement between specialists of the different areas that make part of the research group.

Keywords: Software engineer, image processing and analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A indústria do software vem experimentando um grande crescimento nas últimas décadas. Hoje, o software está presente na vida de praticamente todas as pessoas, seja através do uso de computadores, sistemas de automação comercial e industrial ou software embutido em eletrodomésticos, telefones celulares, etc. As principais conseqüências desse crescimento são o aumento da complexidade do software e as exigências cada vez maiores do mercado (COELHO, 2003). É requerido das empresas de software que os sistemas sejam desenvolvidos com prazo e custo determinados e obedeçam a padrões de qualidade. Para atender essas solicitações, tornou-se necessário investir na metodologia/processo de desenvolvimento de software, já que é cada vez mais evidente a correlação entre a qualidade do produto de software desenvolvido e a metodologia de desenvolvimento adotada (MACHADO, 2000).

Estes problemas afligem a área de software desde sua criação. Pressman os identifica como uma “aflição crônica", e não uma crise pontual (PRESSMAN, 2002). Para administrá-los, a comunidade de engenharia de software, ao longo dos últimos 30 anos, estuda e implementa práticas de desenvolvimento de software bem organizadas e documentadas.

Grande parte do trabalho dos pesquisadores envolvidos com a engenharia de software tem sido descrever e abstrair modelos que descrevem processos de software. Estes modelos permitem que se compreenda o processo de desenvolvimento dentro de um paradigma conhecido. A existência de um modelo é apontada como um dos primeiros passos em direção ao gerenciamento e à melhoria do processo de software (KELLNER, 1989).

A maioria dos modelos existentes é baseada em premissas bastante tradicionais da área: uma divisão discreta das atividades do processo de software, focando em gerenciamento, medidas e registros destas atividades (REIS, 2001).

Uma metodologia de desenvolvimento de software pode ser definida como “um conjunto de atividades, métodos, práticas e transformações que as pessoas empregam para desenvolver e manter software e produtos associados (por exemplo,

plano de projetos, documentos de projeto, projeto de software, código, casos de teste e manual do usuário)", (Ministério da Ciência e Tecnologia, 1999). Uma metodologia tem por objetivo final possibilitar o desenvolvimento de software com qualidade e obedecendo a prazo e orçamento determinados. Através da adoção de uma metodologia, as organizações tentam criar estruturas que facilitem o desenvolvimento e garantam a qualidade do produto, fazendo com que o sucesso de um projeto não dependa apenas do esforço e talento dos membros das equipes de desenvolvimento.

Desse modo, a demanda por processos e metodologias de desenvolvimento de software fez surgir inúmeras alternativas, entre as quais se destacam a RUP[1] (KELLNER, 1989) e (LINDVALL, 2000) e o eXtreme Programming (BECK, 2000). Estas metodologias podem ser adotadas por completo ou parcialmente como processo padrão de uma organização. Porém, são metodologias genéricas que pecam ou por serem muito complexas, de difícil entendimento e gerando um número excessivo de documentos ou por não oferecerem nenhum controle sobre o projeto. Entretanto, uma alternativa é definir uma metodologia específica, que esteja alinhada com os objetivos do projeto, gere um número suficiente de documentação, permita o planejamento contínuo, entendimento dos métodos matemáticos e que possibilite a integração de novos membros na equipe.

A importância do desenvolvimento de uma metodologia própria, ou seja, para desenvolvimento de software em processamento e análise de imagens, está relacionada com a qualidade do software produzido em um menor tempo possível, além de destacar que a eficiência e eficácia do desenvolvimento estão diretamente ligadas à metodologia adotada. Assim, utilizando uma metodologia para desenvolvimento mais adequada ao projeto, pode-se conseguir melhorias no desenvolvimento e na qualidade do produto.

Portanto, no presente trabalho será apresentado e validado uma metodologia de desenvolvimento de software para processamento e análise de imagens, cujo objetivo é garantir que o software construído seja de qualidade, bem documentado e que venha a contribuir para melhorar a velocidade de construção dos sistemas dessa natureza.

---

[1] Rational Unified Process: processo de engenharia de software desenvolvida pela Rational Software Corporation. (http://www.rational.com).

1.1 Estado da Arte

Observa-se que as ciências exatas, que têm forte base em tecnologia, apresentam pesquisas com um ciclo de vida mais iterativo do que comparado a outras ciências onde evoluções ocorrem com menor velocidade (BLOIS, 2004). Na área de engenharia de software, grande parte das pesquisas desenvolvidas é baseada nas abordagens de desenvolvimento de software que vêm evoluindo com o passar dos anos.

Em meados da década de 80, o paradigma estruturado para desenvolvimento de software se destacava, onde a organização de programas era baseada em processos os quais eram vinculados aos problemas a serem resolvidos, com ênfase na modularização de sistemas em termos de subsistemas vinculados a processos. A reutilização dos artefatos gerados (procedimentos e funções) tornava-se complicada, tendo em vista que a solução para um processo do negócio dependia de uma série de artefatos espalhados pelo sistema.

Na década de 80 surgiu o paradigma de orientação a objetos, sendo aplicado em maior escala nos anos 90. As soluções dessa abordagem trouxeram muitas esperanças para a reutilização e produção de componentes. Entretanto, a idéia de que componentes, por si só, seriam uma solução para a reutilização em larga escala, não foi evidenciada na prática (GIMENES, 2005). Essa abordagem fornece muitos recursos para a produção de componentes, mas não alavancou a esperada reutilização em massa. Como indicado por Kruchten (2000 apud ROSSI, 2004, p.100), a arquitetura, o processo e a organização são fatores críticos para a reutilização.

A arquitetura de software é um dos campos em considerável crescimento através dos últimos anos, e promete um contínuo crescimento para a próxima década. A maturidade do projeto arquitetural dentro da disciplina de engenharia de software já é universalmente reconhecida e praticada.

Uma propriedade arquitetural representa uma decisão de projeto relacionada a algum requisito não-funcional (SOMMERVILLE, 2001). A presença de uma determinada propriedade arquitetural pode ser obtida através da utilização de estilos

arquiteturais que possam garantir a preservação dessa propriedade durante o desenvolvimento do sistema (SHAW, 2001).

Na orientação a objetos, um sistema é organizado em termos de classes que encapsulam seus dados e comportamentos. Esta abordagem de desenvolvimento também privilegia a modularização de um sistema, mas em termos de classes (BLOIS, 2004). No entanto, outros benefícios não existentes no paradigma estruturado podem ser citados. Um dos benefícios é o uso de uma linguagem de modelagem unificada Unified Modeling Language (UML), a qual se trata de uma combinação de várias propostas de notações para projeto orientado a objetos, onde modelos de diferentes níveis de abstração e artefatos são propostos para que, de forma combinada, possam dar melhor apoio ao projeto orientado a objetos.

Com o projeto orientado a objetos, tornam-se mais concretos os benefícios para prover a reutilização, em especial através de tecnologias como frameworks (FAYAD, 1999), através do conceito de classes abstratas, padrões (GAMMA et al., 2000), como uma proposta de reutilização de experiências recorrentes de projeto e componentes, pela sua principal característica de ser auto-contido e com interfaces bem definidas (BROWN, 2000), (SAMETINGER, 1997).

Oriundo do surgimento do paradigma orientado a objetos, observou-se que o foco da reutilização vinha ocorrendo em termos de código fonte e, dentro desta linha, várias bibliotecas foram surgindo no mercado, como por exemplo, as bibliotecas para construção de interfaces gráficas. Contudo, a reutilização não é uma abordagem somente aplicável ao contexto de implementação de um sistema, ou seja, é possível reutilizar artefatos de análise, projeto, desde que propostos de forma a prover esta reutilização.

Dentre as tecnologias de reutilização que foram bastante aplicadas no contexto de código fonte destaca-se a de componentes. Embora tal tecnologia tenha sido aplicada inicialmente com este propósito, percebe-se uma forte tendência no uso de componentes (SAMETINGER, 1997), (BROWN, 2000) como artefatos de projeto e implementação, os quais possuem características que privilegiam a reutilização. Nesta área, existem métodos como UML Components (CHEESMAN e DANIELS, 2001), RUP, Kruchten (2000 apud ROSSI, 2004, p.100) e Catalysis[2] (D'SOUZA e WILLS, 1999) que propõem processos para a construção de aplicações

[2] Metodologia não proprietária, dirigida a modelos para a construção de sistemas abertos distribuídos a partir de componentes e frameworks. (http://www.catalysis.org).

baseadas em componentes, alguns deles combinando o uso de frameworks e padrões. No entanto, em geral, estes métodos apresentam certa complexidade para a obtenção de requisitos de uma aplicação e também não deixam claro como ocorre o processo de reutilização dos componentes gerados.

1.2 Definição do Problema

Um desafio constante da área de engenharia de software é melhorar o processo de desenvolvimento de software (FILHO, 2005). Mesmo com a constante evolução dos métodos, técnicas e ferramentas, a entrega de software em prazos e custos estabelecidos nem sempre é conseguida. Segundo Larman (2004), uma das causas desse problema é a falta de formalidade nos modelos de processo propostos nos últimos 30 anos.

Existe hoje a necessidade de desenvolver software de forma mais rápida, mas acima de tudo com qualidade. No âmbito do desenvolvimento de software para processamento e análise de imagens há a necessidade que o desenvolvimento seja feito de forma organizada, seguindo uma metodologia que respeite cada etapa do desenvolvimento e inclua a garantia da qualidade em cada fase.

Dos trabalhos científicos publicados em revistas que tratam sobre processamento e análise de imagens a maioria aborda a resolução de um ou vários problemas relacionados à área, não sendo apresentada qual foi a metodologia utilizada para resolver esses problemas, isso se dá pela inexistência de uma metodologia de desenvolvimento para projetos desse tipo.

O problema encontrado por equipes que desenvolvem software para processamento e análise de imagens é a falta de uma metodologia de desenvolvimento que permita um planejamento contínuo e adaptabilidade às mudanças requeridas, além de organizar, de forma prática, o processo de desenvolvimento, incluindo a etapa necessária para o entendimento do problema.

Outro problema encontrado é a possibilidade das equipes de desenvolvimento estarem geograficamente dispersas. Em parte, é responsável por esta tendência o desejo de aproveita ao máximo as habilidades do pessoal disponível para o projeto, não importando sua localização.

Com intuito de minimizar tais problemas, se faz importante a elaboração de uma metodologia específica o para desenvolvimento de software em processamento e análise de imagens que tenha como objetivo garantir que o software construído seja de qualidade, bem documentado, e que a produtividade do grupo de desenvolvimento seja mantida, independente de sua localização.

1.3 Contribuições

A dependência e demanda crescentes da sociedade em relação à Informática e, em particular, o software, tem ressaltado uma série de problemas relacionados à forma de desenvolvimento de software: alto custo, alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e o produto desenvolvido (DEMARCO, 1995).

A maior parte das metodologias existentes é baseada em premissas bastante tradicionais da área: uma divisão discreta das atividades do processo de software, focando em gerenciamento, medidas e registros destas atividades.

No desenvolvimento de software para processamento e análise de imagens a realidade não é a mesma. Isso ocorre tendo em vista que os sistemas para processamento e análise de imagens fazem uso de algoritmos extremamente complexos, necessitam de mais recursos computacionais e de memória, além da necessidade de adaptação dos métodos matemáticos para o contexto do problema que se está resolvendo.

Sabe-se que a área de processamento e análise de imagens tem atraído grande interesse nas últimas duas décadas. Com a evolução das tecnologias de computação digital, cada vez mais as aplicações que envolvem imagens estão presentes no cotidiano. Essas aplicações podem ser encontradas na astronomia, biologia, medicina nuclear, geografia e em aplicações industriais.

Nesse contexto, fazer uso de uma metodologia que seja específica a esse tipo de desenvolvimento é de grande validade, tendo em vista que cresce cada vez mais a procura por software desse tipo. Com esse intuito, a contribuição da presente dissertação está centrada no desenvolvimento de uma metodologia para construção de software para processamento e análise de imagens.

No contexto onde foi realizado o estudo de caso, as principais contribuições foram a organização do trabalho da equipe como um todo e a etapa de integração dos membros na equipe, a qual permite que os novos integrantes tenham conhecimento sobre o que é o projeto e sobre o estagio atual de desenvolvimento.

1.4 Estrutura da Dissertação

Este trabalho está estruturado da seguinte maneira. No Capítulo 2 são apresentadas as técnicas de engenharia de software aplicadas ao processamento e análise de imagens. As técnicas abordadas são: a arquitetura de software e seus componentes, padrões de projeto como solução para prover a reusabilidade de projeto e código e por fim a linguagem de modelagem unificada - UML, a qual foi utilizada para modelagem da metodologia.

No Capítulo 3, é apresentado o desenvolvimento incremental bem como a qualidade das aplicações fazendo uso desse tipo de desenvolvimento. É abordado também nesse capítulo a metodologia extreme programming (XP) como uma instância de *rational unified process* (RUP).

No Capítulo 4, é apresentada a metodologia proposta para desenvolvimento de software para processamento e análise de imagens – MetImage, sendo abordado em detalhes todas as suas etapas. O Capítulo 5 apresenta o estudo de caso da metodologia MetImage e da metodologia extreme programming em um projeto real de desenvolvimento de software, apresentando os resultados alcançados ao utilizar ambas as metodologias. Por fim, o Capítulo 6 apresenta as conclusões e sugestões para os trabalhos futuros.

2 TÉCNICAS DE ENGENHARIA DE SOFTWARE APLICADAS AO PROCESSAMENTO E ANÁLISE DE IMAGENS

Atingir a satisfação de clientes e usuários através de sistemas de software que supram seus requisitos e que possuam o maior tempo de vida possível é o objetivo de desenvolvedores e pesquisadores da área de engenharia de software. No entanto, a complexidade imposta pelos sistemas atuais exige que técnicas e métodos sejam desenvolvidos e usados para tratar ou gerenciar tal complexidade.

Atualmente, as aplicações de software para processamento e análise de imagens têm crescido no tamanho e na complexidade, fazendo com que seu projeto e construção se tornem uma tarefa bastante difícil. Com isso, deve haver certo cuidado quanto ao seu projeto, isto é, na forma como será implementado, fazer uso de técnicas de engenharia de software é um fator crucial para o sucesso no desenvolvimento desse tipo de sistema. As técnicas de engenharia de software aplicadas ao desenvolvimento de sistemas para processamento e análise de imagens serão discutidas nas subseções seguintes.

2.1 Arquitetura de Software e Seus Componentes

Arquiteturas de software ganharam uma ampla popularidade na última década, tendo sua importância reconhecida na engenharia de software. Considera-se que ela exerça um papel fundamental em lidar com dificuldades inerentes ao desenvolvimento de sistemas de software complexos e de grande porte (CLEMENTS, 1996).

Normalmente, o desenvolvimento de sistemas grandes e complexos, o que é característico no desenvolvimento voltado para imagens, ocorre em camadas (MARCUS VÖLTER e WOLFF, 2002). A vantagem de fazer uso de uma arquitetura organizada em camadas ocorre pelo fato de oferecer um maior entendimento da aplicação e por dividi-la em um conjunto de componentes que interagem entre si

para realizar parte de uma ou várias funcionalidades do sistema. Além de permitir que a complexidade intrínseca dos sistemas de software para processamento e análise de imagem possam ser melhor gerenciados.

Dentre essas camadas as mais utilizadas são: a camada núcleo, a camada lógica de negócio e a de apresentação. A camada núcleo é responsável pelos algoritmos essenciais na área de processamento e análise de imagens. A próxima camada é referenciada como a camada lógica de negócio (MARCUS VÖLTER e WOLFF, 2002), a qual é responsável por abrigar implementação de diversos algoritmos para imagens na forma de componentes de classes. Esses componentes são inter-relacionados entre si com o objetivo de resolver um problema maior a partir de pequenos artefatos de software.

E por fim, a camada de apresentação, onde os componentes são responsáveis por estabelecer a comunicação com as plataformas de execução do software (BLOIS, 2004).

A Figura 2.1, mostra uma visão geral da estrutura de uma arquitetura em camadas.

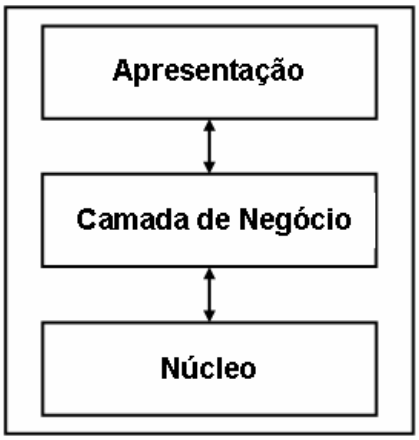

Figura 2.1 - Componentes de uma arquitetura em camadas

O principal objetivo em organizar a arquitetura em camada está em tornar o software flexível em relação as suas funcionalidades, isto é, dividi-lo em blocos de funções similares ao seu nível de abstração (BUSCHMANN, 1996).

A utilização do conceito de arquitetura no desenvolvimento de sistemas para processamento e análise de imagens representa o principal suporte para desenvolver e principalmente manter sistemas de software de alta durabilidade. A arquitetura de software define, através de um alto nível, o sistema em termos de componentes, a interação entre eles e os atributos e funcionalidades de cada componente (SOMMERVILLE, 2001). Fazer uso de uma definição bem clara da arquitetura dá uma perspectiva de todo o sistema e do controle necessário para seu desenvolvimento e gerenciamento da complexidade existe para o desenvolvimento desse tipo de sistema.

As principais vantagens em fazer uso de uma arquitetura bem definida no desenvolvimento de sistemas complexos, o qual é uma das características do desenvolvimento voltado para imagens, a saber são:

- Permite que a estrutura suporte as funcionalidades do sistema, levando em conta o comportamento dinâmico do sistema;
- No nível arquitetural todos os detalhes de implementação são escondidos;
- Permite alto-nível de abstração suficiente para que o sistema possa ser visto como um todo;
- Auxilia no gerenciamento da complexidade do software bem como torna o mesmo flexível em relação à incorporação de novos requisitos do usuário.

A arquitetura de um sistema é uma estrutura ou conjunto de estruturas que incluem componentes de software, as propriedades destes componentes que são visíveis externamente e o relacionamento entre esses componentes. Em processamento e análise de imagens fazer uso de uma arquitetura que permita o desenvolvimento baseado em componentes auxilia na substituição dos componentes obsoletos por outros mais robustos e estáveis, além de permitir que os mesmos sejam reutilizados em outras aplicações.

Segundo Brown (2000), a partir do conhecimento inicial das necessidades do negócio, um método de desenvolvimento baseado em componentes deve prever uma primeira etapa de entendimento do contexto, onde são definidos os requisitos iniciais da aplicação, e modelados os casos de uso que definem os usuários e as atividades por eles desempenhadas, bem como os tipos de negócio, ou seja, a representação dos principais elementos do negócio e seus relacionamentos no domínio. Para o desenvolvimento desta primeira etapa, o conhecimento do domínio e dos sistemas já existente é fator chave.

A próxima etapa envolve a definição da arquitetura, a qual é desenvolvida a partir dos aspectos estáticos e dinâmicos modelados na fase anterior. A seguir, as atividades previstas na fase de definição de arquitetura são apresentadas:

- Modelagem da arquitetura de componentes, onde uma coleção de componentes relacionados são propostos e definidos. A arquitetura de componentes gerada pode organizá-los de acordo com o tipo de serviço oferecido (e.g. infra-estrutura, negócio);
- Modelagem do contexto, para entender o escopo do software que está sendo desenvolvido. Em geral, este entendimento compreende uma especificação mais exata das responsabilidades dos componentes em relação aos casos de uso do domínio;
- Modelagem de interface, para obter um conjunto de interfaces candidatas e descrevê-las em detalhe. Para dar início a esta atividade, sugere-se que o projetista considere os resultados da modelagem de contexto e dos tipos do negócio para eleger o primeiro conjunto de interfaces candidatas;
- Definição de interface, na qual interfaces obtidas como resultados da atividade anterior são descritas em detalhe (CHESSMAN e DANIELS, 2001).

A última etapa prevista num método de desenvolvimento baseado em componentes, segundo Brown (2000), é a proposta de solução para o problema, onde os componentes são efetivamente implementados. Nesta etapa, deve-se viabilizar a comunicação entre componentes gerados com os sistemas já existentes e ainda componentes de terceiros.

Enfim, fazer uso de uma arquitetura que possibilite o desenvolvimento baseado em componentes na construção de software para a manipulação e análise de imagens permite que os seus componentes fiquem independentes permitindo que os mesmos possam ser reutilizados em outras aplicações.

2.2 Padrões de Projeto e Reusabilidade

O processamento e análise de imagens são campos da computação que vêm crescendo continuamente e ocupando espaços de importância nos mais diversos

meios. Na área de visão computacional, o objetivo básico é fazer com que as máquinas possam ver, ou seja, mais do que capturar uma imagem; o desafio é possibilitar que as máquinas sejam capazes de processar e posteriormente analisar as imagens obtidas. No campo de imageamento médico, com o avanço da medicina e da tecnologia empregada na área, a utilização de imagens tornou-se indispensável, seja em um simples exame ou na realização de uma complexa cirurgia feita via computador.

Entretanto, as imagens digitais estão cada vez mais dependentes de um software que gerencie todo o seu processamento ou análise. Nesse contexto, que os padrões de projetos são utilizados.

Os padrões de projeto são formas de construção de software comprovadamente funcionais que garantem legibilidade, fácil manutenção e reutilização de código-fonte no desenvolvimento de algumas aplicações computacionais.

Segundo Gamma et al. (2000), padrões de projeto podem ser utilizados para definir as abstrações necessárias e auxiliar na modelagem do sistema. Ao se deparar com um problema similar a outro já resolvido, o desenvolvedor pode reutilizar a essência de sua solução para resolver o novo problema (BUSCHMANN, 1996). Essa solução consiste em um padrão de projeto.

A principal justificativa para fazer uso de padrões no desenvolvimento de sistemas para processamento e análise de imagens está no fato de que a fase de projeto é uma das fases do processo de desenvolvimento de software que necessita de mais tempo e trabalho para a sua conclusão. Além de poder auxiliar, através de seu uso, a tornar o sistema mais gerenciável no que diz respeito a suas funcionalidades.

Segundo Gamma et al. (2000) , há 23 exemplos de padrões de projeto que procuram atender às necessidades de todos os domínios de aplicação. Contudo esses não são os únicos padrões existentes. Segundo Metsker (2002) existem outros 77 padrões que, juntos, provavelmente caracterizam os 100 mais usuais. Entre esses outros padrões podem-se citar os de Buschmann (BUSCHMANN, 1996), (*Pattern-Oriented Software Architecture* - POSA), que se preocupa em aplicar formas ótimas de construção de software sob vários níveis como o arquitetural, o de projeto e o idiomático.

Os padrões de projeto são catalogados de forma a alcançar uma padronização nas fases de análise e projeto, até atingir um nível em que a nomenclatura dos padrões de projeto seja adotada como linguagem padrão para comunicação dentro das equipes de desenvolvimento.

Esses padrões são agrupados seguindo dois critérios. O primeiro é o propósito, que diferencia os padrões de projeto conforme a sua função. O segundo é o contexto, que separa os padrões de projeto conforme a sua aplicação nos objetos ou nas classes.

Padrões de Projeto, segundo Gamma et al. (2000) são estruturas que ocorrem diversas vezes dentro de seu contexto. Em outras palavras, um padrão de projeto especifica a estrutura e o comportamento de um conjunto de classes em um projeto de software.

Segundo Salviano (1997), padrões de software podem ser definidos através de quatro perspectivas: uma atitude, que representa uma visão humanística do software, uma ênfase onde é identificado e comunicado boas e práticas soluções, um objetivo onde é criado uma literatura e uma forma que é o padrão concreto obtido.

Uma das vantagens de fazer uso de padrões no desenvolvimento de software para processamento e análise de imagem está no fato de que o seu uso auxilia a ocultar a complexidade de algumas soluções e manipula-la de forma mais flexível.

Outra vantagem em fazer uso de padrões para o desenvolvimento desse tipo de sistema está centrada na possibilidade de poder ter os componentes reutilizados para outra aplicação uma vez que cada componente pode encontrar-se fisicamente separado para fins organizacionais, de manutenção e de legibilidade.

Em processamento e análise de imagem fazer uso de padrões como forma de auxiliar o reuso é de extrema importância. Isso ocorre pelo fato de que os sistemas que gerenciam esse tipo de uso são provedores de algoritmos extremamente complexos e com isso fazer uso de uma técnica que permita reutilizar a solução de um problema em um outro auxilia na redução de tempo, aumento da produtividade e entendimento do algoritmo implementado.

Mesmo fazendo uso de padrões de projeto como uma forma de prover o reuso, a sua implementação em grande escala tem enfrentado grandes dificuldades para atingir os resultados relativos ao aumento da produtividade, à melhoria da qualidade e à redução dos custos do processo de desenvolvimento.

Na maioria das vezes, as organizações têm a idéia de que a reutilização é um problema de aquisição de tecnologia (BOEHM, 1999), o que não é realidade. A reutilização de software é um problema de transição de tecnologia, não bastando adquirir um sistema de biblioteca de componentes de software para fazer este processo ser implementado.

Com isso, outros aspectos devem ser considerados na implementação da reutilização no processo de desenvolvimento de sistemas. Um desses aspectos é a necessidade de mudanças no próprio processo de desenvolvimento, para fornecer suporte e promover a reutilização de software.

Além disso, a estrutura da organização deve passar por mudanças para que seja disponibilizada uma infra-estrutura para permitir o desenvolvimento com reutilização. Mas antes de implementar a reutilização no processo de desenvolvimento é importante saber suas vantagens e desvantagens.

Na literatura são apontados muitos aspectos vantajosos decorrentes da reutilização de software (MCCLURE, 1997). Entre essas vantagens, pode-se citar:

- Redução do custo de desenvolvimento;
- Melhoria da qualidade do software;
- Aumento da produtividade do software;
- Melhoria da interoperabilidade entre sistemas de software;
- Compartilhamento do conhecimento dos sistemas e da forma de construção;
- Compartilhamento dos componentes de software de modo mais formal.

Entre as desvantagens, Mcclure (1997) classifica o custo de construção de componentes de software reutilizável como sendo a principal desvantagem, tendo em vista que o mesmo é caro e consome mais tempo para construção.

É importante ressaltar que o sucesso da implantação do reuso depende de três fatores básicos: atitude, ferramentas e técnicas (METSKER, 2002). A atitude que deve ser tomada envolve a conscientização da equipe da necessidade de reuso. Muitas vezes, por comodidade ou por receio de utilizar programas alheios, prefere-se desenvolver um componente novamente. Essa é uma atitude que deveria ser ponderada nas empresas. Ferramentas de auxílio ao reuso são fundamentais: elas auxiliam na busca, armazenagem e recuperação dos componentes. Técnicas para reuso também são necessárias para seu sucesso: regras para projeto e codificação

de componentes reusáveis aumentam a facilidade de entendimento e conseqüentemente o reuso de um componente.

Enfim, fazer uso de padrões no desenvolvimento de sistemas voltados para processamento e análise de imagens propicia formas comprovadamente ótima de gerenciar a complexidade dos componentes com o objetivo de torná-los modulares e flexíveis para fins de reuso.

2.3 Linguagem de Modelagem Unificada – UML

A linguagem UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson numa tentativa de definir uma forma padronizada e unificada de modelagem para resolver vários problemas. Atualmente ela é a notação gráfica mais amplamente utilizada para a modelagem de sistemas orientados a objetos, pois sua linguagem é visual e rica em recursos necessários para formalizar, documentar e demonstrar a criação, comportamento e estrutura dos objetos (ARMOUR, 2003).

A razão pela qual fazer uso de UML no contexto de processamento e análise de imagens ocorre pelo fato da mesma facilitar a comunicação de certos conceitos mais claramente do que outras linguagens. Além de permitir, por exemplo, saber a capacidade do hardware de um equipamento (quantidade de memória, tipo de placa de vídeo), informações que são necessárias quando se deseja, por exemplo, realizar um processamento tridimensional intensivo.

A UML é uma linguagem de modelagem e não uma metodologia, ou seja, não tem noções do processo, pois um método consiste de uma linguagem de modelagem e de um processo. A linguagem de modelagem é a notação (principalmente gráfica) utilizada por métodos para expressar projetos. O processo é a sugestão de quais passos a serem seguidos na elaboração de um projeto.

Isto significa que se podem utilizar os conceitos de UML, sem ficar preso a um processo padrão, permitindo que o desenvolvedor estabeleça o seu próprio processo de acordo com suas necessidades, em função do tipo de software a ser desenvolvido (tempo real, sistema de informações, produto desktop), o tamanho da equipe a ser utilizada, o nível de formalismo exigido e a facilidade de comunicação

desejada, o que lhe confere a flexibilidade para ser usada com todos os tipos de processos e em todo o ciclo de desenvolvimento de software.

A UML é uma linguagem de modelagem visual que utiliza vários tipos de diagramas para auxiliar o analista e o projetista a documentar parte ou todo o processo de software. Os diagramas são uma apresentação gráfica de uma coleção de elementos de modelagem (símbolos gráficos) freqüentemente relacionados por arcos e vértices (relacionamentos), que ilustram partes distintas do software (TIBERTI, 2003).

Alguns desses diagramas são: diagrama de casos de uso, diagrama de classe, diagrama de colaboração, diagrama de seqüência e diagrama de implantação. Mas nem todos esses diagramas necessitam ser utilizados no desenvolvimento de um software, cabe ao responsável por cada atividade escolher os diagramas a serem utilizados.

Essa linguagem é uma tentativa de padronizar a modelagem orientada a objetos de uma forma que qualquer sistema seja qual for o tipo, possa ser modelado corretamente, com consistência, fácil de comunicar com outras aplicações, simples de ser atualizado e principalmente compreensível.

A principal motivação para fazer uso de UML no desenvolvimento de sistemas para processamento e análise de imagens é que permite à definição das classes utilizadas e como elas se relacionam entre si na solução do problema, além de ser uma linguagem de fácil interpretação, permitindo entender a complexidade que esse tipo de sistema apresenta.

UML é uma ferramenta essencial não apenas para documentar um sistema, mas também para desenvolver uma solução antes do processo de codificação. Por exemplo, no desenvolvimento de componentes de software para processamento e análise de imagens, pode-se ter um ou mais modelos estruturais dependendo do nível de complexidade do sistema desenvolvido, vindo a modularizar a solução do problema de forma mais legível.

Sendo atualmente a notação gráfica mais amplamente utilizada para a modelagem de sistemas complexos e orientados a objetos. Isso ocorre porque sua linguagem é visual e rica em recursos necessários para formalizar, documentar e demonstrar a criação, comportamento e estrutura dos objetos (ENGELS e GROENEWEGEN, 2000).

2.3.1 Fases do Desenvolvimento Usando UML

Um processo de desenvolvimento classifica em atividades as tarefas realizadas durante a construção de um sistema de software. Segundo Bezerra (2003), existem vários processos de desenvolvimento propostos, no qual cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento. Entretanto podem-se distinguir atividades que, com uma ou outra modificação, são comuns à maioria dos processos existentes. A saber, as fases de desenvolvimento são:

- Levantamento dos Requisitos: esta fase corresponde à etapa de compreensão do problema aplicada ao desenvolvimento de software. O principal objetivo do levantamento de requisitos é que usuários desenvolvedores tenham a mesma visão do problema a ser resolvido. Nessa etapa, os desenvolvedores, juntamente com os clientes, tentam levantar e definir as necessidades dos futuros usuários do sistema a ser desenvolvido (MACIASZEK, 2000).
- Análise dos Requisitos: nesta fase, os analistas realizam um estudo detalhado dos requisitos levantados na etapa anterior. A partir desse estudo, são construídos modelos para representar o sistema a ser construído (BEZERRA, 2003). Nessa fase, o foco de interesse é tentar construir uma estratégia de solução sem se preocupar com a maneira como essa estratégia será realizada.
- Projeto: nessa fase, determina-se "como" o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes (BEZERRA, 2003). Esta fase produz uma descrição computacional do que o software deve fazer e deve ser coerente com a descrição feita na análise. O projeto consiste de duas atividades principais: projeto da arquitetura (também conhecido como projeto de alto nível) e projeto detalhado (também conhecido como projeto de baixo nível). Em um processo de desenvolvimento orientado a objetos, o projeto da arquitetura consiste em distribuir as classes de objetos relacionadas do sistema em subsistemas e seus componentes (BEZERRA, 2003). No projeto detalhado, são

modeladas as colaborações entre os objetos de cada módulo com o objetivo de realizar as funcionalidades do módulo. Também é realizado o projeto da interface com o usuário e o projeto de banco de dados.

- Implementação: na fase de implementação, as classes provenientes do projeto são codificadas, ou seja, ocorre a tradução da descrição computacional da fase de projeto em código executável através do uso de uma ou mais linguagens de programação (BEZERRA, 2003).
- Testes: diversas atividades de testes são realizadas para verificação do sistema construído, levando-se em conta a especificação feita na fase de projeto. Um sistema normalmente é rodado em testes de unidade, integração e aceitação. Os testes de unidade são para classes individuais ou grupo de classes. Os testes de integração são aplicados já usando as classes e componentes integrados para se confirmar se as classes estão cooperando uma com as outras como especificado nos modelos. Os testes de aceitação observam o sistema como uma "caixa preta" e verificam se o sistema funciona como especificado.
- Implantação: o sistema é empacotado, distribuído e instalado no ambiente do usuário. Os manuais do sistema são escritos, os usuários são treinados para utilizar o sistema corretamente. Em alguns casos, aqui também ocorre a migração de sistemas de software e de dados preexistentes (BEZERRA, 2003).

3 DESENVOLVIMENTO INCREMENTAL

O desenvolvimento de software evoluiu com o passar do tempo. Em muitos contextos, os requisitos do sistema mudam durante o desenvolvimento, impossibilitando um desenvolvimento linear até o produto final; os prazos de entrega são tão curtos que impossibilitam à entrega do produto completo, sendo necessária a produção de uma versão limitada para o cumprimento do prazo; e somente os requisitos de uma versão básica estão bem definidos, faltando à definição de maiores detalhes do produto.

Com o intuito de solucionar os problemas citados acima que o modelo de desenvolvimento incremental foi desenvolvido. Foram combinadas as características positivas dos modelos lineares, que propõe a divisão do desenvolvimento em estágios sucessivos (BOEHM, 1999), e prototipação, que propõe o desenvolvimento de um protótipo inicialmente, para que o cliente possa especificar melhor os requisitos do sistema. Esse modelo é visto como uma alternativa para a vasta gama de métodos para desenvolvimento de software. No modelo incremental o desenvolvimento é dividido em etapas, denominadas "incrementos", que produzirão incrementalmente o sistema, até a sua versão final.

No desenvolvimento de software para processamento e análise de imagens fazer uso desse tipo de desenvolvimento auxilia a baixar o risco de o projeto falhar completamente. Isso ocorre porque se acontece um grande erro, apenas o último incremento é descartado, vindo com isso reduzir o tempo gasto com o desenvolvimento, sem falar que a construção de sistemas complexos, típico na área de processamento de análise de imagens, realizada em um sistema menor é sempre menos arriscada que a construção de um grande sistema.

Mas é importante lembrar que para fazer uso do desenvolvimento incremental na construção de sistemas complexos, uma intensa fase de análise a cada incremento é extremamente importante, pois é imprescindível que o sistema seja bem projetado para que possa acomodar as mudanças posteriores.

3.1 Qualidade das Aplicações no Desenvolvimento Incremental Usando Componentes

Grande parte da população mundial depende de aplicações de software, seja para realizar suas atividades profissionais ou de lazer. As empresas de software estão preocupadas em alcançar patamares cada vez maiores de qualidade e de produtividade, para enfrentarem a competitividade que também é cada vez maior. Com o aumento no tamanho e na complexidade, os produtos de software estão se tornando artefatos estratégicos e críticos. Logo, uma atenção especial tem sido voltada ao desenvolvimento e à manutenção de produtos de software com qualidade dentro de restrições de tempo e de recursos estabelecidos nos projetos (ROCHA et al., 2001).

Adicionalmente aos problemas de projetos cancelados e com orçamentos excedidos, está a entrega de produtos não confiáveis e com baixas taxas de qualidade. Confiabilidade, disponibilidade, cronograma e custos são geralmente os conceitos de mais alta prioridade para os usuários.

Aplicações de software são produtos complexos, difíceis de desenvolver e testar. Muito freqüentemente, produtos de software apresentam comportamento inesperado e indesejado, que podem causar muitos problemas (FUGGETA, 2000).

A garantia da qualidade dessas aplicações é a principal atividade com a qual uma equipe de desenvolvimento, manutenção e engenharia devem se preocupar para entregar um produto confiável a seus usuários.

Em processamento e análise de imagens, onde esses tipos de sistemas estão presentes nos mais diversos segmentos, como por exemplo, na área médica, onde tem a função de auxiliar os profissionais da saúde a fazer uma avaliação mais criteriosa sobre determinada doença, a garantia da qualidade é um fator de extrema importância.

Atributos de qualidade rigorosos devem ser satisfeitos por este tipo de software já que falhas nos mesmos podem causar danos irreparáveis. Para Cagnin (2005), essa atividade deve ser conduzida em todas as fases do processo de software para que os defeitos introduzidos sejam eliminados na fase em que

surgiram, a fim de evitar o aumento dos custos da remoção desses em fases posteriores da sua criação.

Como forma de auxiliar na garantia da qualidade para esse tipo de sistema, se torna necessário fazer uso de mecanismos que auxiliem na resolução e no entendimento do problema. Entre esses mecanismos pode-se citar o desenvolvimento baseado em componentes. Esse tipo de desenvolvimento soluciona os problemas quebrando-os em porções menores e implementando as soluções como componentes independentes.

Cheesman (2001) enfatiza o desenvolvimento baseado em componentes como uma técnica para o desenvolvimento de produtos de software complexos, que priorizam a redução dos custos e o aumento da produtividade e da qualidade.

Da mesma forma que no desenvolvimento tradicional de software, no desenvolvimento incremental, vários fatores influenciam na qualidade dos componentes quando utilizados em sistemas para processamento e análise de imagens:

- Qualidade das metodologias de desenvolvimento: a qualidade da metodologia utilizada para desenvolver software para imagens é um fator muito importante. Faz-se necessário utilizar uma metodologia que auxilie o desenvolvimento desde o princípio garantindo que o produto seja confiável, eficaz, eficiente e de fácil manutenção.
- Arquitetura do software, estilos arquiteturais, modelos e framework de componentes: em sistemas com um alto grau de complexidade, como em processamento e análise de imagens, a arquitetura do sistema tem uma função muito importante. Utilizar uma arquitetura baseada em componentes torna o software flexível quanto suas funcionalidades e permite uma fácil manutenção, além de propiciar o reuso. Além de que, muitas das propriedades de qualidade da aplicação final são, em boa parte, resultantes da arquitetura utilizada.
- Qualidade dos componentes, documentação e artefatos em geral: Sametinger define como atributos de qualidade os aspectos não funcionais de um componente, tais como segurança, performance e confiabilidade. Para sistemas complexos, como é o caso dos sistemas para processamento e análise de imagens, esses aspectos não funcionais são de extrema importância, tendo em vista que as imagens resultantes muitas

vezes auxiliam a diagnosticar ou constatar algum tipo de problema nas mais diversas áreas. Sametinger (1997), define também algumas propriedades que deveriam ser analisadas na avaliação da qualidade de um dado componente, são elas: realização de testes, clareza conceitual, definição precisa para o grau de acoplamento e coesão.

Dessa forma qualidade não pode ser entendida como perfeição. Qualidade de software é algo factível, relativo, substancialmente dinâmico e evolutivo, adequando-se ao nível dos objetivos a serem atingidos (BELCHIOR, 1997). Alcançar graus elevados de qualidade custa caro; assim, o importante é atingir o nível desejado pelos usuários (BOEGH et al., 1993).

3.2 Usando o Rational Unified Process (RUP) no Desenvolvimento Incremental

O Rational Unified Process (RUP) é uma metodologia de desenvolvimento de software criada pela Rational Software Corporation. O RUP é um framework de processo iterativo e incremental que abrange as melhores práticas do desenvolvimento de software de mercado e tem sido largamente utilizado em projetos de software (SOUZA, 2005).

Segundo Kruchten (2000 apud ROSSI, 2004, p.100) o RUP é uma instância mais específica do Processo Unificado e as melhores práticas de processo de software incorporadas neste processo são:

- Desenvolvimento iterativo;
- Gerência de requisitos;
- Utilização de arquitetura e componentes;
- Modelagem visual através da UML;
- Qualidade de processo e de produto;
- Gerência de configuração e versões;
- Casos de uso dirigindo muitos aspectos de desenvolvimento;
- Modelo de processo de desenvolvimento que pode ser adaptado e estendido para as características da organização;

- Necessidade de ferramentas de desenvolvimento de software para fornecer suporte ao processo.

Conforme apresentado na Figura 3.1, o RUP possui duas dimensões. O eixo horizontal, segundo Kruchten (2000 apud ROSSI, 2004, p.101), representa o tempo e mostra como os componentes do ciclo de vida do processo são desdobrados através das suas fases. Esta representação descreve os aspectos dinâmicos do processo como ele ordena e expressa em termos de ciclos, fases, iterações, e os pontos de verificação, sendo que, dentro de cada fase, gerentes ou projetistas podem dividir o trabalho em duas ou mais iterações e cada fase termina com um ponto de verificação.

No eixo vertical representa o fluxo de trabalho do processo, agrupando as atividades logicamente pela natureza Kruchten (2000 apud ROSSI, 2004, p.101). Este processo, segundo Rossi (2004), abrange um ciclo de vida de desenvolvimento completo, consistindo basicamente de modelagem de negócio, gerência de requisitos, análise e projeto, implementação, teste e distribuição.

É importante ressaltar que podem existir outros elementos no ciclo de vida, e que a cada elemento é apoiado por ferramentas e guias de processo descrevendo cada um deles e suas aplicações.

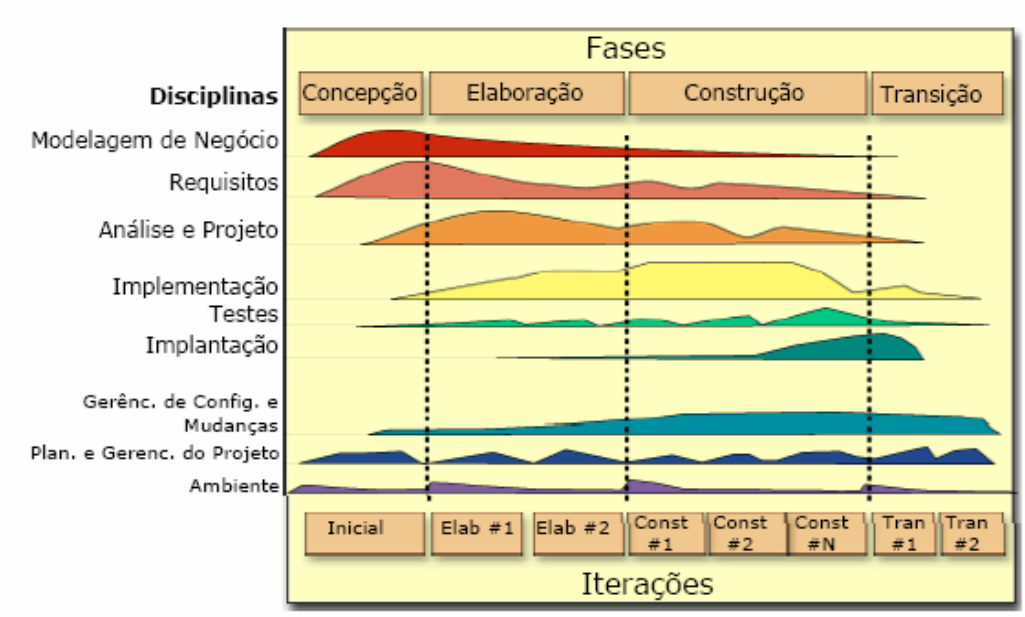

Figura 3.1 - Ciclo de vida de desenvolvimento do RUP

As disciplinas fundamentais do processo de desenvolvimento de software também estão presentes na estrutura do RUP. A disciplina de Requisitos é responsável por estabelecer e manter concordância com os clientes e outros envolvidos sobre o que o sistema deve fazer, oferecer aos desenvolvedores do sistema uma compreensão melhor dos requisitos e definir as fronteiras do sistema.

Segundo Souza (2005), a disciplina de Análise e Projeto, por sua vez, visa transformar os requisitos em um projeto do sistema e desenvolver a arquitetura. O processo é baseado em caso de uso e desenvolve a análise e projeto através de Realizações de Casos de Uso. A finalidade da disciplina de Implementação é implementar classes e objetos, testar e integrar os resultados produzidos. A disciplina de Testes atua em vários aspectos como uma provedora de serviços para as outras disciplinas, enfatizando principalmente a avaliação da qualidade do produto.

Mesmo sabendo que o RUP é uma metodologia voltada para o desenvolvimento de sistemas comerciais, desenvolvimento por contrato e desenvolvimento interno, nada o impede de ser utilizado para o desenvolvimento de software para processamento e análise de imagens.

As vantagens em fazer uso desse processo na área de processamento e análise de imagens ocorrem pelo fato do RUP ser baseado em boas práticas de desenvolvimento, como o desenvolvimento incremental. Esse tipo de desenvolvimento leva em consideração que os requisitos podem mudar, algo que é bem comum na área de imagens, além de permitir que a integração do sistema seja feita progressivamente, vindo com isso diminuir a complexidade do sistema e facilitar o seu entendimento.

Outra vantagem diz respeito à arquitetura baseada em componentes. Como foi abordado no Capítulo 2, ao utilizar esse tipo de arquitetura em sistemas para processamento e análise de imagens permite que a estrutura suporte as funcionalidades do sistema, levando em conta o comportamento dinâmico do sistema, auxilia no gerenciamento da complexidade do software bem como torna o mesmo flexível em relação a novos requisitos do usuário. Além de permitir que os mesmos possam ser reutilizados em outras aplicações.

Entretanto, o RUP possui algumas desvantagens. Primeiro por ser um processo muito complexo necessita de certo tempo para o seu entendimento, tendo em vista que o mesmo precisa ser adaptado às características de cada projeto e

segundo por ser uma metodologia considerada pesada por gerar um número excessivo de documentação.

3.2.1 Elementos do RUP

O RUP é um framework composto por vários tipos de elementos que, em conjunto, formam o processo completo (BOEHM, 1999), a saber são:

- Atividades: definem o trabalho que será feito. Uma atividade é constituída por um conjunto de passos, que definem como a atividade deve ser realizada. Cada atividade é realizada por um ou mais papéis e possui artefatos de entrada e saída.
- Papéis: são os responsáveis por realizar as atividades. Um papel não está associado a uma pessoa específica. Um papel pode ser desempenhado por mais de uma pessoa e uma pessoa pode desempenhar mais de um papel.
- Artefatos: são os produtos gerados pelas atividades. Podem ser documentos, modelos ou elementos de software, como programas ou bibliotecas de componentes.
- Disciplinas: são agrupamentos de atividades interrelacionadas. Uma disciplina tem como função determinar a ordem em que as atividades serão executadas.
- Fases: O desenvolvimento utilizando o RUP é feito em fases, que indicam a ênfase do projeto em um dado instante. O RUP possui quatro fases:
    - Concepção: definição clara do que é o produto a ser desenvolvido, para isso, devem ser estabelecidos o escopo do sistema, uma visão inicial dos principais casos de uso, as condições limites e o critério de aceitação.
    - Elaboração: elaboração da especificação detalhada de mais casos de uso do produto e da arquitetura do sistema. Esta fase pode ser executada em uma ou mais iterações, dependendo do escopo, tamanho, e o nível de domínio sobre o assunto do projeto.

- Construção: processo de produção do produto, ou seja, implementação do software, integração e testes dos componentes e avaliação da versão dos produtos em relação aos critérios de aceitação. O resultado desta fase é o produto que incorpora todos os casos de usos, definidos pelo cliente e pela gerência, para fazer parte desta versão.
- Transição: conjunto de atividades que conduzem o produto obtido ao ambiente de produção, compreendendo, por exemplo, manufatura, treinamento de usuário, suporte e correção de erros encontrados depois da entrega.

Uma fase é dividida em iterações, que são intervalos de tempo definidos onde uma parte do sistema é desenvolvida. Na Figura 3.1, é apresentado à relação entre fases, iterações e disciplinas, mostrando a concentração das atividades de cada disciplina ao longo das fases do processo.

Pode-se ver, por exemplo, que as atividades da disciplina Análise e Projeto recebem maior ênfase na fase de Elaboração enquanto as atividades disciplina Implementação estão concentradas na fase de Construção e recebem pouca ênfase na fase de Concepção. As atividades da disciplina Planejamento e Gerenciamento estão, em comparação com as atividades das demais disciplinas, distribuídas de forma mais homogênea ao longo de todo o projeto.

3.3 Extreme Programming (XP) como uma Instância de RUP

Atualmente RUP e XP são as duas principais metodologias de desenvolvimento de sistemas sendo adotadas ou cogitadas por equipes de desenvolvimento de software. Ambas têm muitas características comuns, como requerer seqüências de atividades ou fases onde pessoas desempenham papéis bem definidos para gerar artefatos que são encaminhados ao cliente e/ou realimentam o ciclo (SMITH, 2001). Entretanto, distingue-se bastante em uma análise mais detalhada, o que torna cada uma mais indicada para um determinado perfil de projeto (SMITH, 2001).

Segundo Smith (2001), o *Rational Unified Process* vem sendo refinado ao longo de anos, coletando as melhores práticas de desenvolvimento de software e agrupando-as em um framework que basicamente visa: desenvolver software iterativamente, gerenciar requisitos, usar arquiteturas baseadas em componentes, modelar o software visualmente, verificar a qualidade de software e controlar mudanças no sistema. RUP permite ser configurado de modo a se ajustar às necessidades individuais de cada projeto ou características de equipes de trabalho.

Em contrapartida, o eXtreme Programming aparece como uma alternativa mais leve para times de tamanho pequeno e médio porte, que desenvolvem software em um contexto de requisitos vagos e rapidamente modificados (BECK, 2000). Pela característica de simplicidade que esta técnica apresenta, poucos artefatos, papéis e atividades são definidos. O eXtreme Programming tem obtido reconhecimento através de sucessos alcançados por pequenos projetos em contextos de grandes mudanças de requisitos.

Entretanto, a leveza deste método e o foco em código tornam outros aspectos importantes de um projeto, como a gestão de requisitos e a construção da arquitetura, que são pouco enfatizados.

Dentre as principais diferenças da XP em relação a outras metodologias estão:

- Feedback constante;
- Abordagem incremental;
- A comunicação entre as pessoas é encorajada.

Apesar de não encontrar nenhuma referência na literatura sobre o uso dessas metodologias no desenvolvimento de sistemas para processamento e análise de imagens, nada impede que as mesmas sejam utilizadas para esse tipo de desenvolvimento.

Entretanto, para alguns autores como Williams e Cockburn (2003), existem circunstâncias adequadas para o uso dessas metodologias. Para eles as circunstâncias adequadas são: 1) o sistema a ser criado não é de missão crítica; essa é uma das características dos sistemas para processamento e análise de imagens, por exemplo, na área médica, onde as imagens possuem informações pertinentes sobre determinada doença. 2) os requisitos mudam com freqüência e 3) a equipe de desenvolvimento é pequena, eficiente e situada na mesma localidade.

A maioria das regras da XP causa polêmica à primeira vista e muitas não fazem sentido se aplicadas isoladamente. Para Bona (2002, p.41) “é a sinergia de seu conjunto que sustenta o sucesso de XP, encabeçando uma verdadeira revolução no desenvolvimento de software”.

A XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente de desenvolvimento de software para os seus seguidores, que são conduzidos por quatro valores: comunicação, simplicidade, feedback e coragem (BECK, 2000).

A finalidade do princípio de comunicação é manter o melhor relacionamento possível entre clientes, desenvolvedores e gerente de projeto, preferindo conversas pessoais a outros meios de comunicação. A simplicidade visa permitir a criação de código simples que não deve possuir funções desnecessárias.

Outra idéia importante da simplicidade é procurar implementar apenas requisitos atuais, evitando-se adicionar funcionalidades que podem ser importantes no futuro. A aposta da XP é fazer algo simples hoje e pagar um pouco mais amanhã, para fazer modificações necessárias, do que implementar algo complicado hoje que talvez não venha a ser usado, sempre considerando que requisitos são mutáveis.

A prática do feedback significa que o programador terá informações constantes do código e do cliente. A informação do código é dada pelos testes constantes, que indicam os erros tanto individuais quanto do software integrado. Em relação ao cliente, o feedback constante significa que ele terá freqüentemente uma parte do software totalmente funcional para avaliar. O cliente então, constantemente, sugere novas características e informações aos desenvolvedores. Desta forma, a tendência é que o produto final esteja de acordo com as expectativas reais do cliente.

As doze práticas que norteiam o desenvolvimento em XP são citadas e descritas resumidamente a seguir (BECK, 2000):

- Jogo de Planejamento: consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto. A XP baseia-se em requisitos atuais para desenvolvimento de software, não em requisitos futuros.
- Versões Pequenas: coloca rapidamente o sistema em funcionamento, liberando novas versões em pequenos intervalos de tempo. Com isso, o

cliente pode verificar se o sistema atende os requisitos inicialmente definidos.

- Metáfora: são as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.
- Projeto Simples: o programa desenvolvido pelo método XP deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros. Eventuais requisitos futuros devem ser adicionados assim que eles realmente existirem. Esta forma de raciocínio se opõem ao "implemente para hoje e projete para amanhã".
- Testes Constantes: a XP focaliza a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os testes.
- Programação em Pares: a implementação do código é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador. O desenvolvedor que está com o controle do teclado e do mouse implementa o código, enquanto o outro observa continuamente o trabalho que está sendo feito, procurando identificar erros sintáticos e semânticos e pensando estrategicamente em como melhorar o código que está sendo implementado.
- Refatoração: focaliza o aperfeiçoamento do projeto do software e está presente em todo o desenvolvimento. A refatoração deve ser feita apenas quando é necessário, ou seja, quando um desenvolvedor da dupla, ou os dois, percebe que é possível simplificar o módulo atual sem perder nenhuma funcionalidade.
- Propriedade Coletiva: o código do projeto pertence a todos os membros da equipe. Isto significa que qualquer pessoa que percebe que pode adicionar valor a um código, mesmo que ele próprio não o tenha desenvolvido, pode fazê-lo, desde que faça a bateria de testes necessária. Isto é possível porque na XP todos são responsáveis pelo software inteiro. Uma grande vantagem desta prática é que, caso um membro da equipe deixe o projeto antes do fim, a equipe consegue continuar o projeto com poucas dificuldades, pois todos conhecem todas as partes do software, mesmo que não seja de forma detalhada.

- Integração Contínua: interagir e construir o sistema de software várias vezes por dia, mantendo os programadores em sintonia, além de possibilitar processos rápidos. Integrar apenas um conjunto de modificações de cada vez é uma prática que funciona bem porque fica óbvio quem deve fazer as correções quando os testes falham: a última equipe que integrou código novo ao software. Esta prática é facilitada com o uso de apenas uma máquina de integração, que deve ter livre acesso a todos os membros da equipe.
- 40 Horas de Trabalho Semanal: a XP assume que não se deve fazer horas extras constantemente. Caso seja necessário trabalhar mais de 40 horas pela segunda semana consecutiva, existe um problema sério no projeto que deve ser resolvido não com aumento de horas trabalhadas, mas com melhor planejamento, por exemplo.
- Cliente Presente: é fundamental a participação do cliente durante todo o desenvolvimento do projeto. O cliente deve estar sempre disponível para sanar todas as dúvidas de requisitos, evitando atrasos e até mesmo construções erradas. Uma idéia interessante é manter o cliente como parte integrante da equipe de desenvolvimento.
- Código Padrão: padronização na arquitetura do código, para que este possa ser compartilhado entre todos os programadores.

Segundo Beck (2000), XP é descrito sob os seguintes aspectos: ciclo de desenvolvimento, histórias, versões, iteração, tarefa e teste. O ciclo de desenvolvimento de XP pode variar de anos a dias. O cliente decide o que estará na próxima versão do sistema, selecionando as características mais importantes do sistema a partir de todas as estórias possíveis. O cliente pode selecionar um conjunto de estórias e o programador calcula a data final de entrega da versão do sistema (CAGNIN, 2005).

O objetivo de cada iteração em XP é colocar em produção novas histórias que são implementadas, testadas e que tornam prontas para serem utilizadas pelos clientes. Segundo Cagnin (2005), o processo inicia com um plano que termina as histórias para serem implementadas e as divide em tarefas para que as equipes possam implementá-las. O programador responsável pela equipe primeiro encontra um parceiro, existindo qualquer dúvida sobre o escopo ou sobre a abordagem de implementação, os parceiros tem uma reunião curta (15 minutos) com o cliente e/ou

desenvolvedor mais experiente sobre a parte do código que será manipulada durante a implementação.

Enquanto a equipe está implementando, o cliente está especificando os testes funcionais, baseando-se no que poderia convencê-lo de como as histórias de uma iteração deve se comportar para estar operando corretamente (CAGNIN, 2005). No final da iteração todos os testes de unidade e funcionais devem ser executados e a equipe deve estar pronta para a próxima iteração.

Contudo, alguns pontos são bastante questionados em relação à XP. Entre esses pontos o de maior impacto diz respeito à documentação. Esse fator é bastante questionado em sistemas que foram desenvolvidos a partir de um projeto XP, isso ocorre devido à falta de documentação gerada pela metodologia.

Em processamento e análise de imagens, por ser uma área muito complexa e necessitar de uma equipe multidisciplinar, a documentação é um fator importante, pois além de permitir auxiliar na manutenção, auxilia na inserção de novas funcionalidades do sistema e até mesmo no reuso, além de otimizar a comunicação e facilitar o entendimento dos algoritmos complexos.

4 METIMAGE: UMA METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE PARA O PROCESSAMENTO E ANÁLISE DE IMAGENS

O sucesso do desenvolvimento de software está intrinsecamente relacionado à metodologia de desenvolvimento utilizada. Mesmo com a constante evolução dos métodos, ferramentas e técnicas, a entrega de software em prazos e custos estabelecidos nem sempre é conseguida (FUGGETA, 2000).

Sem uma metodologia de desenvolvimento, os riscos são maiores uma vez que existe maior dependência das pessoas envolvidas e a verificação do andamento se torna uma atividade complexa e imprecisa.

Ao desenvolver uma metodologia específica para a área de processamento e análise de imagens, levaram-se em consideração alguns fatores que são importantes para a área, os quais serão descritos a seguir:

- Adaptabilidade e Planejamento Contínuo: os requisitos são levantados aos poucos e o planejamento é contínuo, para que a adaptação às mudanças possa ocorrer. Essa adaptabilidade às mudanças podem ser nos requisitos do sistema, tecnologias de implementação utilizadas e equipes de projeto dentro de período de desenvolvimento.
- Equipe Multidisciplinar: o desenvolvimento de sistemas para processamento e análise de imagens necessita de uma equipe multidisciplinar, ou seja, deve possuir especialistas na área de aplicação do software, além de especialistas em processamento e análise de imagens e métodos numéricos.
- Complexidade dos Algoritmos Implementados: essa complexidade é decorrente do fato de que normalmente as técnicas de processamento e análise de imagens são baseadas em métodos numéricos que permitem descrever quantitativamente imagens das mais diversas origens (ALBUQUERQUE, 2005).

Sabendo que uma metodologia de desenvolvimento define o fluxo de trabalho, as atividades, os artefatos e as regras para os envolvidos no trabalho, Conallen (2000) define quais os objetivos que uma metodologia deve possuir:

- Prover direção sobre a ordem das atividades do time;
- Especificar quais artefatos devem ser desenvolvidos;
- Direcionar as tarefas de desenvolvedores e do time como um todo; e
- Oferecer critérios para monitorar e mensurar os produtos do projeto e das atividades.

A importância da utilização de uma metodologia para o desenvolvimento de software é evidente, tendo em vista que cada vez mais aumentam as necessidades de desenvolver software de forma mais rápida, mas acima de tudo com qualidade. A metodologia desenvolvida, assim como seus métodos, ferramentas e procedimentos são discutidos nas subseções seguintes.

4.1 Ciclo de Vida da Metodologia

A engenharia de software é ainda muito incipiente se for levar em consideração o pouco tempo da sua existência. Provavelmente a forma como os sistemas são construídos atualmente sofrerá intensa mudança nas próximas décadas (PRESSMAN, 2002).

Atualmente existem definições de metodologias que em termos de complexidade e burocracia, pregam o extremo. Estes extremos são conhecidos como metodologias leves (a mais conhecida é a XP - é considerada leve por oferecer menor controle, maior velocidade e maior simplicidade) e as metodologias pesadas (entre elas RUP - por oferecer maior controle e conseqüentemente exige a criação de artefatos, o que a torna mais burocrática e rígida), as quais foram descritas nas seções anteriores.

A presente dissertação objetiva conciliar práticas desses dois tipos de definições de metodologia para se obter um ponto de equilíbrio entre a rigidez das metodologias pesadas e o menor controle das metodologias leves, criando uma alternativa de desenvolvimento que seja adequada para a construção de software em processamento e análise de imagens.

Em uma metodologia para desenvolver software em processamento e análise de imagens deve-se levar em consideração alguns fatores, como o tamanho do projeto, o qual pode ser medido de várias formas, através do tamanho da equipe, em razão do tempo de duração, etc. Alta complexidade dos sistemas desenvolvidos, equipe geograficamente dispersa, rotatividade das pessoas e a multidisciplinaridade da equipe de desenvolvimento.

Outro item importante é o ciclo de vida da metodologia. No caso da metodologia proposta, o ciclo de vida possui as fases de iniciação, elaboração, construção e transição. A Figura 4.1, mostra as fases do ciclo de vida da metodologia MetImage.

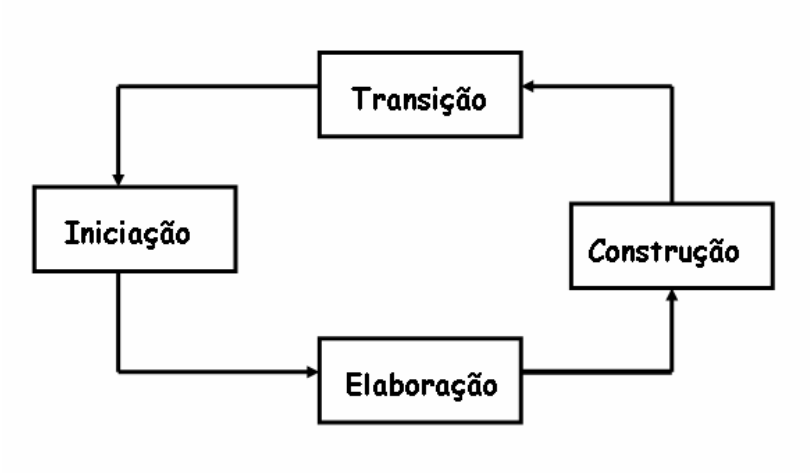

Figura 4.1 - Ciclo de vida

4.1.1 Iniciação

Essa fase tem como objetivo prover o planejamento necessário para o desenvolvimento do sistema, levando em consideração quem vão ser os usuários e que os requisitos podem ser alterados constantemente. Nessa fase os seguintes pontos são definidos: definição do problema, levantamento dos requisitos, definição das tecnologias, definição da equipe e, por fim, definição de release, conforme detalhamento a seguir:

1) **Definição do Problema**

Essa definição é passada aos integrantes do grupo na forma de um seminário, devendo ser explorados os pontos importantes e principais do problema.

2) Levantamento dos Requisitos

Tendo em vista que a natureza dos problemas para essa área é complexa é necessário que os requisitos sejam levantados aos poucos e constantemente.

3) Definição das Tecnologias

Nessa etapa as tecnologias bem como a linguagem de programação, devem ser escolhidas levando em consideração as necessidades para a solução do problema. A metodologia proposta levou em consideração alguns critérios na hora de propor a linguagem de programação Java em conjunto com a API JAI[3]. Esses critérios serão apresentados a seguir:

- Modularidade: Java é uma linguagem modular pois permite a construção de unidades de compilação o que é válido para organizar grandes sistemas.
- Orientação a Objetos: Java é uma linguagem totalmente orientada a objetos, o que permite a herança e a reutilização de código de forma dinâmica e estática.
- Reusabilidade: Java permite o reuso não só de classe, mas também de pacotes inteiros.
- Portabilidade: essa é uma das principais características da linguagem Java. Os programas escritos nessa linguagem podem ser movidos de plataforma operacional sem que haja qualquer modificação em seu código fonte.
- Legibilidade: por Java ser considerada uma linguagem alto nível o seu código possui uma tendência a se apresentar legível e, dessa forma, torna-se um dos pré-requisitos para a fácil manutenção de código (WELFER, 2005).
- Disponibilidade de ferramentas: existe uma vasta gama de ferramentas que possibilitam a construção de código em Java.

A justificativa para escolha dessa linguagem se dá pela simplicidade de implementação, aliada a uma série de recursos disponíveis através de bibliotecas, e a independência de plataforma, uma vez que o código compilado pode ser

[3] Java Advanced Image – Página Oficial: http://java.sun.com/products/java-media/jai.

executado em qualquer plataforma que possua uma máquina virtual Java instalada (ECKEL, 2002).

Para implementação do código voltada para o processamento e análise de imagens API JAI, fornece um grande número de operações, que permitem as mais diversas formas de manipulação de imagens, por exemplo, segmentação, operadores aritméticos, geração de histograma, filtros, adição de imagens, entre outros.

4) Definição da Equipe

No que tange a definição da equipe deve-se levar em consideração a experiência técnica e no domínio da aplicação. A cada entrada de um novo membro na equipe o mesmo passa pela fase de adaptação e aprendizagem. Primeiramente o novo integrante é apresentado para o projeto.

Essa apresentação se da na forma de um seminário de integração, onde um membro da equipe é escalado para fazer essa apresentação, sendo apresentado o que é o projeto, os objetivos, quem são os usuários, como está sendo desenvolvido o sistema e quais as tecnologias que estão sendo utilizadas no processo de desenvolvimento.

Após a essa fase é necessário que o novo membro fique integrado sobre o atual estágio de desenvolvimento. Isso ocorre através da documentação gerada pelo grupo em todas as etapas do processo de desenvolvimento. Posteriormente é levado em consideração a experiência técnica e o conhecimento sobre o domínio da aplicação do novo membro. Caso o novo membro não domine as tecnologias utilizadas pela equipe e os conceitos principais sobre processamento e análise de imagens, o mesmo necessita de um período de aprendizagem. Esse período é determinado pelo gerente de projeto, o qual leva em consideração a experiência do novo membro para determinar o tempo de aprendizagem. Após todo esse ciclo, que é mostrado na Figura 4.2, o novo membro está apto para integrar a equipe de desenvolvimento.

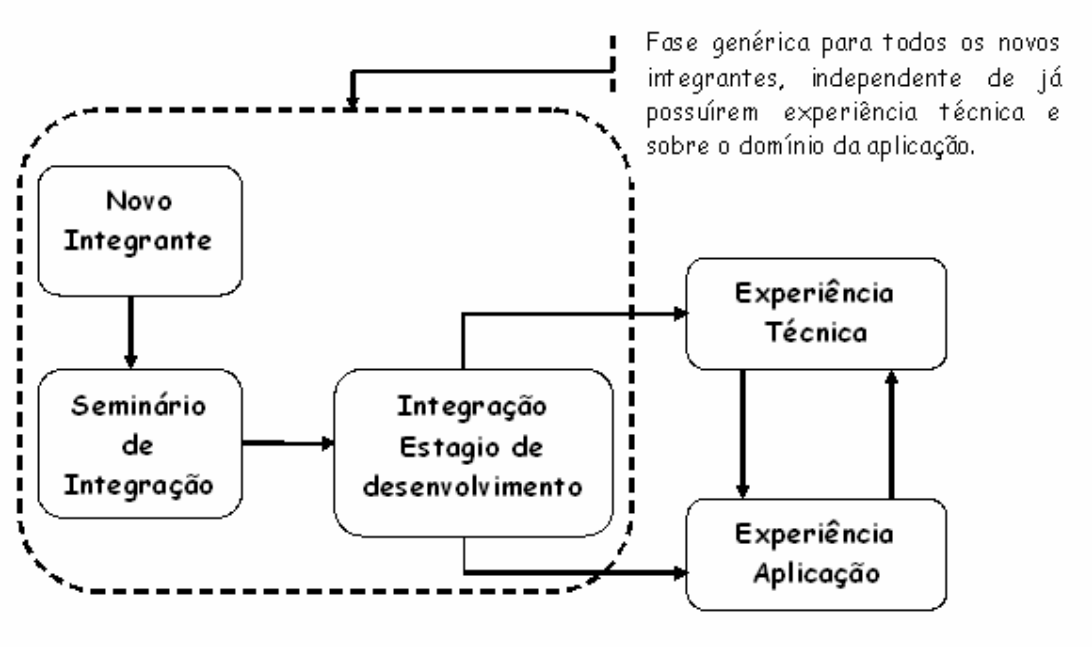

Figura 4.2 - Fase de conhecimento e aprendizagem

É importante ressaltar que essas etapas são necessárias para qualquer novo integrante na equipe, independente de ser desenvolvedor ou testador.

5) Definição de Release

Nessa etapa deve haver o planejamento das versões do sistema que serão liberadas. Deve prever versões menores do sistema para que possa ser verificado se o sistema atende aos requisitos definidos.

4.1.2 Elaboração

Fase de definição da arquitetura. A metodologia proposta sugere que a arquitetura do sistema seja baseada em componentes e que a mesma seja organizada em camadas. Esse tipo de organização em camadas é muito característico no desenvolvimento voltado para imagens, dentre essas, as mais utilizadas são: a camada central, a camada da aplicação e a de apresentação (MARCUS VÖLTER e WOLFF, 2002).

A principal motivação para fazer uso desse tipo de arquitetura é que através dela se obtém modularidade, oferecendo uma separação clara entre as partes da

aplicação, facilitando com isso a manutenção e principalmente o reuso, como foi visto no Capítulo 2.

4.1.3 Construção

Nessa etapa se desenvolve o produto. Além da codificação, também é produzido a documentação e os casos de testes. Nessa fase alguns pontos são decididos:

1) Codificação

Nessa fase é realizada a tradução das necessidades anteriormente traçadas na forma de um seminário para codificação do software que será gerado. Para que essa atividade seja realizada com qualidade, alguns pontos devem ser obedecidos.

- Código Coletivo e Padronizado: qualquer programador pode adicionar novas funcionalidades, corrigir erros ou refazer qualquer parte do código do sistema, desde que garanta a integridade no repositório. Após concluídas as alterações, os testes unitários devem verificar o correto funcionamento da nova versão. Outro item importante é a padronização do código, a presente metodologia de desenvolvimento recomenda a adoção de um padrão desde o início do projeto. Esse padrão é definido pelos desenvolvedores, preferencialmente padrões utilizados pela comunidade da linguagem em desenvolvimento.
- Integração Contínua: ao invés de os programadores trabalharem em paralelo, e somente próximo de uma liberação existir o esforço de integração, o proposto nesse trabalho é que sejam realizadas integrações contínuas. Essa integração é realizada pelos próprios programadores, os quais devem estar em constante comunicação um com os outros, pois assim, eles sabem o que pode ser reutilizado, ou o que pode ser compartilhado. Ferramentas de controle de versões podem ser úteis neste momento. O controle de versão deve ser feito por um único membro da equipe, o qual recebe o código dos demais desenvolvedores e o integra no sistema, ficando a seu cargo avisar os demais membros das atualizações realizadas.

2) Documentação

Tendo em vista que dentro de um projeto de desenvolvimento existe alta rotatividade da equipe, a documentação não pode estar confiada nas pessoas que integram o projeto. Nesse contexto, a documentação deve ser realizada em todas as fases do projeto, desde o levantamento dos requisitos até a implementação.

A documentação gerada na fase de construção ocorre no formato do Javadoc[4], na forma de um relatório de implementação e do relatório de testes. No relatório de implementação deve conter um resumo do que foi desenvolvido, sua funcionalidade, os métodos matemáticos utilizados e o diagrama de classe. A documentação gerada na fase de Iniciação e Elaboração é o relatório do projeto, que deve conter uma descrição geral do projeto e a arquitetura do sistema. Os modelos de documentos gerados pela MetImage encontram-se na seção de Apêndices. É importante ressaltar que a documentação deve ser padronizada pela equipe de desenvolvimento.

3) Testes

A atividade de teste é extremamente importante para garantir a integridade do sistema. No contexto da metodologia proposta os testes serão realizados pela equipe de testes, a qual não está envolvida no desenvolvimento do sistema. Para todo código implementado deve coexistir com um teste automatizado, assim garantindo o pleno funcionamento da nova funcionalidade.

Os testes realizados pela equipe testadora serão os testes de unidade, em que o testador escreve os códigos para verificar se as unidades do sistema funcionam corretamente, e os testes de aceitação os quais tem como objetivo verificar se o software funciona da maneira esperada pelo usuário, ou conforme a especificação.

4.1.4 Transição

Nessa fase é realizado, caso seja necessário, o treinamento dos usuários externos e a manutenção do produto final.

[4] Originária do próprio JDK e permite a documentação interna do código. (http://java.sun.com/products/jdk/javadoc/index.html).

4.2 Papéis

Os papéis podem ser visualizados como funções exercidas ao longo do processo de desenvolvimento. Um membro da equipe pode assumir mais de um papel em um projeto, mas para cada um dos momentos existirão objetivos distintos. Segue a descrição dos papéis:

Gerente de Projeto: pessoa responsável pelos assuntos administrativos do projeto, ou seja, é responsável pela definição do planejamento e controle do projeto, mantendo um forte relacionamento com os usuários para que o mesmo participe das atividades do desenvolvimento, sendo de sua responsabilidade extrair as necessidades junto ao usuário e especificar estas necessidades de maneira compreensível aos demais participantes do processo;

Desenvolvedor: responsável por codificar e documentar o sistema;

Testador: pessoa responsável em garantir a qualidade do sistema através dos testes de unidade e dos testes de aceitação.

Usuário Externo: usuário final ou pessoa que possua visão necessária para o esclarecimento de dúvidas quanto às necessidades a serem atendidas pelo sistema, que também deve ser capaz de homologar as funcionalidades implementadas.

Usuário Interno: pessoa que também possui visão necessária para o esclarecimento de dúvidas, mas que utiliza o que está sendo desenvolvido para fins didáticos e acadêmicos. Esse tipo de usuário são os integrantes da própria equipe de desenvolvimento.

4.3 Atividades e Artefatos

As atividades definem o trabalho que será feito, sendo constituída por um conjunto de passos que definem como as atividades devem ser realizadas. No momento em que as atividades são realizadas, estas criam, utilizam e/ou alteram artefatos, ou seja, são os produtos gerados pelas atividades, que podem ser documentos, modelos ou elementos de software.

Para a definição das atividades, bem como os artefatos, levou-se em consideração o RUP, tendo em vista que o mesmo é considerado uma metodologia que abrange as melhores práticas de desenvolvimento, apesar de ser uma metodologia mais burocrática, como foi especificada no capítulo anterior.

Na metodologia de desenvolvimento proposta, cada etapa do ciclo de vida gera atividades e artefatos que, a saber, são:

1. **Iniciação**

Atividades**:**

- Descrição Geral do Projeto: é o detalhamento de uma visão ampla do sistema. Nessa etapa são definidos os objetivos, propósitos, riscos do sistema e a linha a ser seguida no projeto, ou seja, a equipe de construção, atividades que deverão ser realizadas por cada integrante da equipe e cronograma para início e término das atividades.
- Análise Geral dos Requisitos: é a descrição geral dos requisitos do sistema. Essa etapa tem como objetivo conhecer melhor o escopo do trabalho, tendo o usuário envolvido desde o princípio.

Artefatos:

- Visão do Projeto: descreve uma visão ampla do projeto, nesse artefato os objetivos, propósitos e riscos de projeto devem ser apresentados, além de todo o planejamento.
- Validação dos Requisitos: validação dos requisitos com o usuário do sistema.

2. **Elaboração**

Atividades:

- Especificação do Sistema: essa atividade é responsável pela definição da arquitetura e a elaboração da documentação de análise do sistema. Esta documentação é utilizada pelo projetista para realizar a especificação do que deve ser implementado e também como uma fonte de referência para quando existirem necessidades de iterações do sistema.

Artefatos:

- Especificação do Sistema: contém o detalhamento da arquitetura escolhida e da análise do sistema.

3. **Construção**

Atividades:

- Aprendizagem: etapa necessária para o entendimento dos métodos matemáticos necessários para a implementação.
- Implementação: criação do código do sistema.
- Documentação: elaboração da documentação.
- Teste de Unidade: essa atividade é responsável pela criação das classes de testes do sistema.
- Teste de Aceitação: essa atividade é responsável por realizar os testes de aceitação do usuário.

Artefatos:

- Capacitação: esse artefato possui como resultado o entendimento dos métodos matemáticos. Sendo essa etapa de extrema importância para o entendimento e implementação das funcionalidades.
- Código: código-fonte do sistema.
- Documentação: esse artefato gera como resultado a documentação do que foi implementado no artefato anterior. Essa documentação ocorre na forma do Javadoc e relatório de implementação.
- Relatório do Teste: gera o relatório contendo quais foram as classes testadas.
- Relatório de Aceitação: nessa fase as não conformidades deverão ser registradas e designadas pelo gerente de projeto.

4. Transição

Atividade:

- Liberação: envio da versão do sistema para o usuário.

Artefato:

- Liberação: gera a versão final do sistema devidamente testado e caso seja necessário é realizado o treinamento para os usuários externos do sistema.

A Figura 4.3 mostra os responsáveis pelas atividades e artefatos de cada etapa do ciclo de desenvolvimento da metodologia, sendo importante ressaltar que a etapa de construção é realizada para cada funcionalidade desenvolvida do sistema.

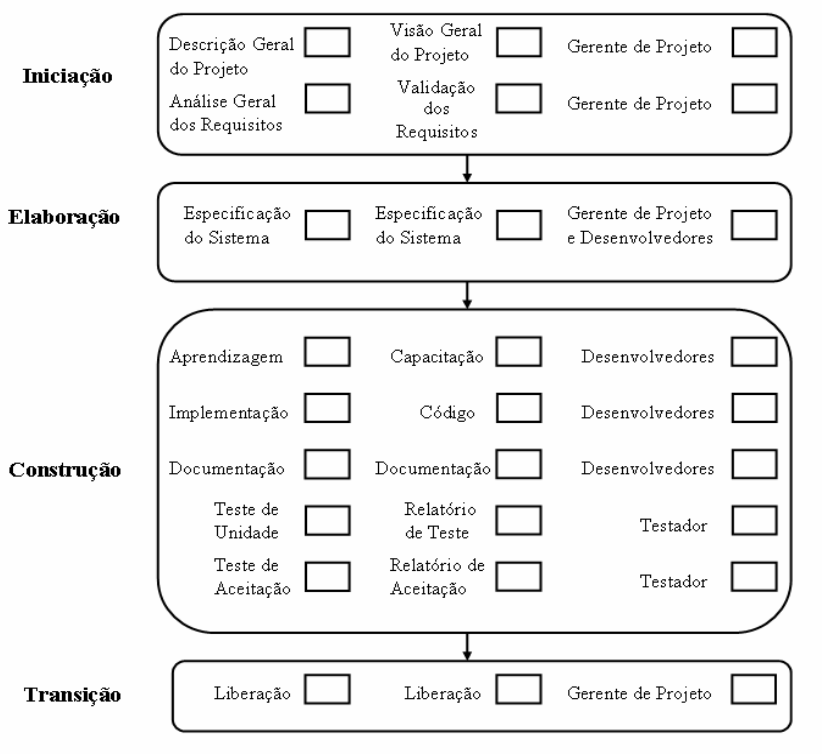

Figura 4.3 - Ciclo detalhado da metodologia

4.4 Gerenciamento do Projeto

Durante todo o processo de desenvolvimento existe um fluxo de suporte responsável pelo projeto. Esta atividade precisa ser realizada ao longo de todo o projeto, sendo de responsabilidade do gerente de projeto. O gerente tem como responsabilidade verificar a real situação do andamento das atividades, transmitir essa visão aos integrantes da equipe e quando necessário replanejar os rumos do projeto, adequando-o a uma nova realidade diferente da que foi planejada inicialmente.

A verificação do andamento das atividades bem como o cumprimento de metas é realizada através de reuniões como sugere a metodologia de desenvolvimento ágil XP. A XP sugere que as reuniões sejam diárias, mas tendo em vista o elevado grau de complexidade dos algoritmos implementados para esse tipo de sistema, as reuniões foram adaptadas para reuniões semanais.

Essas reuniões têm como objetivo apresentar a toda equipe a real situação das atividades, as dificuldades encontradas durante o processo de desenvolvimento e estabelecer as próximas metas que devem ser cumpridas.

Tendo em vista a multidisciplinaridade necessária para desenvolver software para processamento e análise de imagens, a presente metodologia prevê como forma de otimizar a comunicação e facilitar o entendimento entre especialistas de áreas diferentes a utilização do relatório de implementação que deve ser feito na fase de documentação do sistema.

5 ESTUDO DE CASO

Neste capítulo é apresentada a aplicação da metodologia Extreme Programming (XP) e da metodologia proposta neste trabalho, MetImage. A aplicação dessas metodologias ocorreu em projetos reais de desenvolvimento e manutenção de software, com o objetivo de avaliar o impacto da solução proposta. Esta aplicação ocorreu na forma de um estudo de caso, pois resultados e análises foram feitos sobre as metodologias utilizadas.

5.1 Aplicação da eXtreme Programming - XP

A aplicação da metodologia XP foi realizada no contexto de um grupo de pesquisa em um projeto de desenvolvimento e manutenção de software para processamento e análise de imagens, com o objetivo de assegurar sua qualidade e tornar o processo de desenvolvimento mais ágil e flexível. Nesse projeto trabalha-se na implementação da ferramenta Arthemis, a qual foi inicialmente desenvolvida[5] dentro do Grupo de Processamento de Informação Multimídia PIGS da Universidade Federal de Santa Maria.

O grupo surgiu no ano de 2002 como resultado do crescimento das atividades de processamento e análise de imagens e da diversificação das atividades de pesquisa dentro da UFSM. O grupo conta com uma equipe multidisciplinar composta por 20 pessoas das áreas de computação, física, saúde e engenharia florestal.

A ferramenta Arthemis é implementada utilizando a linguagem de programação Java, a API Swing[6] (Application Programming Interface) e Java Advanced Image[7], em conjunto com o desenvolvimento baseado em componentes e padrões de projeto. Sua funcionalidade é voltada para implementar métodos de

[5] Desenvolvida por Daniel Welfer [Welfer, 2005], Gabrielle Dias Freitas [Freitas, 2004], Grasiela Peccini [Peccini, 2004] e Marcos Cordeiro d'Ornellas.}, inicialmente com o nome de StoneAge.
[6] Java Foundation Classes (JFC/Swing): http://java.sun.com/products/jfc.
[7] Página Oficial de Java Advanced Image: http://java.sun.com/products/java-media/jai.

processamento e análise de imagens em microscopia quantitativa e anatomopatologia, contemplando as seguintes áreas de atuação: processamento e filtragem de imagens para melhoramento na qualidade da análise, classificação automática de imagens indexadas por conteúdo, desenvolvimento de software para processamento de imagens e imageamento médico.

Os usuários da ferramenta Arthemis são os profissionais da patologia clínica e profissionais da área da saúde que trabalham com imagens obtidas via microscópio, além dos professores, os quais usam essa ferramenta para fins didáticos. Esses profissionais possuem experiência no domínio da aplicação por serem especialistas na área de microscopia, vindo a utilizar a ferramenta em questão para auxiliar na melhoria do diagnóstico anatomopatológico e na análise micro estrutural de materiais.

5.1.1 Práticas Implementadas

- Padrão de Codificação: tendo em vista que o grupo não possuía nenhum padrão para codificação, o mesmo ficou definido na primeira reunião de implantação da metodologia XP. O grupo possuía um padrão para agregar novos componentes na ferramenta Arthemis, mas o mesmo não estava sendo usado pelos atuais desenvolvedores.
- Projeto Simples: o sistema foi projetado da maneira mais simples possível, utilizando arquitetura em três camadas, desenvolvimento baseado em componentes e padrões de projeto, vindo com isso a facilitar a inclusão de novas funcionalidades no sistema.
- Refactoring: mesmo possuindo um projeto simples, alguns desenvolvedores fizeram mudanças no código implementado, isso se fez necessário para melhorar o entendimento do que tinha sido implementado.
- Testes Automáticos: Tendo em vista que a garantia da qualidade do software é uma das principais atividades com a qual uma equipe de desenvolvimento deve se preocupar para entregar um produto confiável a seus usuários e que o grupo PIGS não utilizava nenhuma forma de controle da garantia da qualidade, essa atividade foi incluída e conduzida

em todas as fases do processo de desenvolvimento do software. Os testes realizados[8] foram os de unidade, os quais tinham como objetivo verificar se cada unidade do que estava sendo desenvolvido funcionava da maneira esperada. Para realizar esses testes utilizou-se a ferramenta JUnit[9].

- Programação em Pares: Esta prática não foi adotada como sugere XP: toda a equipe trabalhar todo o tempo em pares, a programação pareada foi aplicada a somente alguns integrantes do grupo, aqueles integrantes que estavam desenvolvendo partes semelhantes do Arthemis. Isso ocorreu tendo em vista que houve resistência por parte de alguns integrantes do grupo de pesquisa em trabalharem em pares.
- Cliente no Local: tendo em vista que os clientes se localizam na mesma cidade, estes estavam sempre em contato com os desenvolvedores e demais membros da equipe do projeto. Sendo esse fator importante, tendo em vista que facilita o entendimento do problema, garante a aceitação do produto final e auxilia na detecção prévia de problemas.
- Jogo de Planejamento: o planejamento detalhado do projeto foi levado como base está pratica de XP, onde o cliente solicitou as funcionalidades desejadas através de seminários e não através de estórias como sugere a XP. As reuniões diárias foram substituídas por reuniões semanais de aproximadamente 30 minutos. Essa mudança ocorreu devido à complexidade do sistema desenvolvido, com isso a necessidade de um tempo maior para desenvolver cada funcionalidade. Os objetivos das reuniões eram verificar quais foram as dificuldades encontradas pelos desenvolvedores, bem como a definição das próximas metas e atualizar a equipe sobre o andamento do trabalho.
- Propriedade Coletiva: Todos integrantes da equipe tinham liberdade para alterar qualquer arquivo que compõe o software, desde que fosse necessário.

[8] A experiência do grupo em utilizar essa prática dentro do contexto da ferramenta Arthemis gerou um trabalho publicado no Journal of Computer Science – ISSN: 1807-4545.
[9] JUnit é um framework horizontal de código aberto, desenvolvido por Kent Beck e Erick Gamma, com suporte à criação de testes automatizados em Java.

5.1.2 Práticas não Implementadas

- 40 h Semanais: XP prega que os membros da equipe não devem trabalhar mais do que 40 hs semanais. Por isso, essa prática não foi implementada tendo em vista que os desenvolvedores do grupo são alunos de graduação e pós-graduação os quais além das responsabilidades dentro do grupo possuem responsabilidade dentro dos cursos aos quais pertencem.
- Releases Pequenos: tendo em vista o alto grau de complexidade das funcionalidades implementadas pelos desenvolvedores do grupo a entrega de releases pequenos não foi possível.
- Integração Contínua: o grupo não fazia uso de nenhum tipo de ferramenta para auxiliar na integração e nenhuma pessoa era responsável por essa integração, dessa forma, cada desenvolvedor desenvolvia uma funcionalidade e a mesma ficava apenas disponível na sua versão.
- Metáforas: O projeto utiliza o paradigma de programação orientado a objetos, o que torna o projeto de software perto da realidade do cliente. Devido a isto, o uso de metáforas não foi adotado no projeto.

5.1.3 Pontos Relevantes sobre XP no Escopo do Projeto

Com a aplicação da metodologia XP pode-se constatar que essa metodologia é adequada em projetos onde os clientes não sabem exatamente por onde iniciar o desenvolvimento e a probabilidade de mudarem de idéia é grande. Em projetos onde é necessário o planejamento contínuo, equipe multidisciplinar e principalmente, existe complexidade dos algoritmos implementados, essa metodologia não é adequada.

Isso ocorre porque o grupo desenvolve funcionalidades as quais estão baseadas em métodos matemáticos que permitem descrever quantitativamente

imagens das mais diversas origens. Com isso, faz-se necessário que para cada funcionalidade desenvolvida ocorra o aprendizado e entendimento das fórmulas que são utilizadas. Essa etapa de entendimento e aprendizagem, que não é prevista por essa metodologia, é importante, tendo em vista que o processamento e a análise de imagens é uma técnica extremamente dependente do problema que se quer resolver.

A metodologia XP nega muitas práticas que foram tentadas por décadas, algumas consideradas eficientes como à documentação e o levantamento de requisitos. Os requisitos não são construídos para limitar os desenvolvedores, mas uma fase na qual o problema é continuamente refletido e refinado, até que uma solução coerente e simples seja tomada.

Isto não significa dizer que se deva passar tempo excessivo com requisitos e com planejamento, especialmente se eles mudam com freqüência. Porém, são sempre boas vantagens você ter uma idéia geral do problema a ser resolvido e um possível caminho para a solução.

Outro fator levado em consideração refere-se à documentação. Pouca ou nenhuma documentação significa que o conhecimento do projeto está confiado nas pessoas que integram a equipe. Este recurso não é confiável, vindo a ser um problema a cada entrada ou saída de um membro na equipe, porém esse problema pode ser facilmente resolvido se existir documentação e que a mesma seja corretamente mantida.

No contexto de um grupo de pesquisa, onde integrantes da equipe de desenvolvimento possuem um ciclo dentro do grupo, se faz necessário que tudo o que for feito dentro do projeto seja documentado, isso faz com que os novos integrantes não percam tempo tentando entender o que foi feito por outro integrante.

5.2 Aplicação da MetImage no Contexto de um Grupo de Pesquisa

A metodologia proposta foi aplicada a um projeto real de desenvolvimento e manutenção de software, cujo objetivo foi de avaliar os resultados gerados após a sua utilização. Sua aplicação ocorreu dentro do grupo de pesquisa - Grupo de

Processamento de Informação Multimídia PIGS no projeto em andamento da ferramenta Arthemis.

Ao aplicar a MetImage levou-se em consideração que no início do projeto o grupo não utilizava nenhuma metodologia de desenvolvimento, vindo a adotar mais tarde a metodologia eXtreme Programming. Por isso as etapas de Iniciação e Elaboração, sugeridas pela MetImage, já haviam sido definidas. Sendo que a etapa de Iniciação foi realizada a partir do momento em que o grupo teve o projeto[10] aprovado.

5.2.1 Etapa Elaboração

Levando-se em consideração a necessidade de implementar um software que seja flexível em relação as suas funcionalidades, a arquitetura[11] da ferramenta Arthemis foi concebida utilizando o desenvolvimento em camadas. Onde a camada central é responsável por implementar um conjunto de funcionalidades básicas para processamento de imagens.

A camada de aplicação consiste nos componentes de software responsáveis por resolver diversos problemas envolvendo imagens e por fim, a última camada que é constituída pela interface gráfica, onde é permitido que os usuários apliquem essas operações em imagens, de uma forma simples e intuitiva (WELFER, 2005).

A Figura 5.1, extraída de Welfer (2005), mostra a estrutura do desenvolvimento em camadas da ferramenta Arthemis.

---

[10] Com a aprovação do Edital CT-Info/MCT/CNPq 031/2004 pela coordenação do grupo PIGS o Arthemis será utilizado para dar suporte à implementação da proposta intitulada: *"Ferramentas de Processamento e Análise de Imagens para Microscopia Quantitativa Aplicada a Anatomopatologia e Caracterização Microestrutural de Materiais"*. (http://www.cnpq.br/resultadosjulgamento/edital_312004_ctinfo.htm)

[11] Arquitetura do sistema desenvolvida por Daniel Welfer [Welfer, 2005].

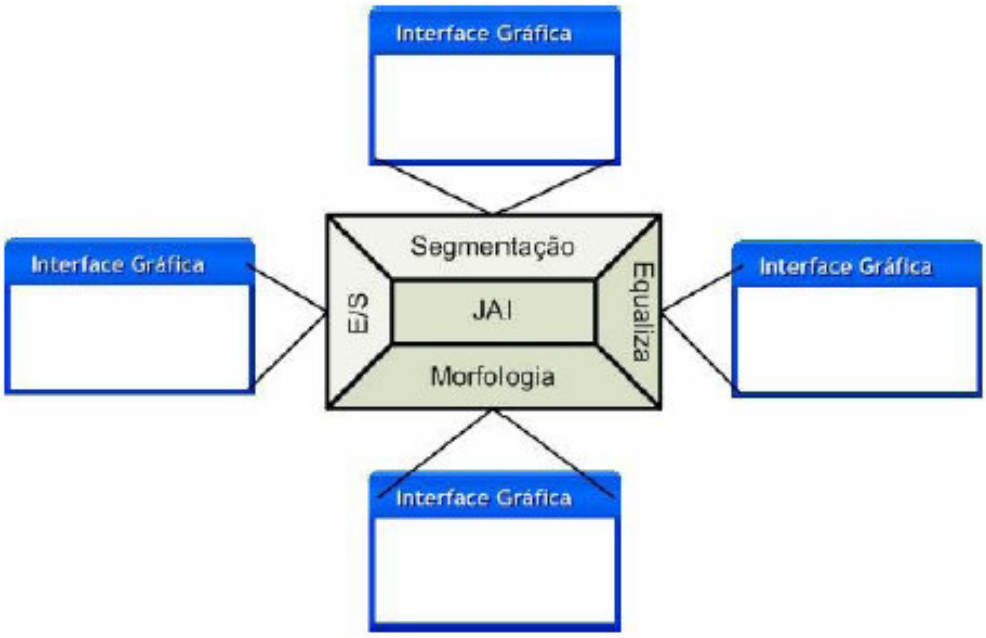

Figura 5.1 - Estrutura em camadas da ferramenta Arthemis

No desenvolvimento da arquitetura do software Arthemis foi utilizado o padrão arquitetural Layer descrito por Buschmann (BUSCHMANN, 1996). A escolha por esse padrão ocorreu pelo fato de que ele permite modelar uma aplicação, intuitivamente, sob camadas onde cada camada representa um domínio na aplicação.

5.2.2 Etapa Construção

Na etapa de construção o grupo passou a adotar algumas padronizações para o desenvolvimento. Essas padronizações foram definidas em conjunto com toda a equipe. A padronização definida pela equipe foi a padronização de código e a padronização adiciona componente. A padronização de código é a forma padronizada de colocar a descrição da função que cada método executa. Esta descrição deve ser incluída no topo do arquivo e possui o formato que é mostrado na Figura 5.2.

```
* Projeto:
* <declaração da classe; nome, descrição breve>
*
* Intenção:
* <descrição do objetivo da classe; como funciona o método,
*  descrição em alto nível>
*
* Hierarquia:
* <descrição da hierarquia da classe>
*
* Fontes de entrada:
*(<posição no vetor>): <classe do objeto> - <descrição do uso>
*
* Parâmetros de entrada:
*(<posição no vetor>): <classe do objeto> - <descrição do uso>
*
* Fontes ou Parâmetros de saída:
*(<posição no vetor>): <classe do objeto> - <descrição do uso>
*
* Autor: <nome do autor>
* Contato: <e-mail para contato>
* Data: <data da ultima modificação>
```

Figura 5.2 - Estrutura padronização de código

A padronização adiciona componente é a forma padronizada que o grupo adotou para a criação de um novo componente (método), o qual deve seguir uma hierarquia de classe onde cada novo componente deve ser incluído dentro do pacote que se encaixe sua funcionalidade. Sendo permitido a criação de um novo pacote caso o novo método não possa ser incluído em nenhum dos pacotes existentes. A Figura 5.3 mostra um exemplo dessa padronização.

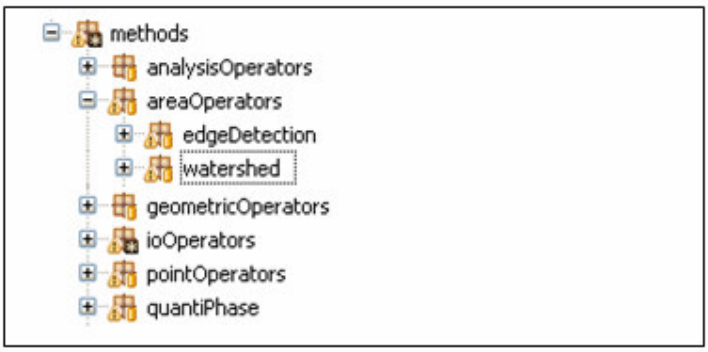

Figura 5.3 - Estrutura padronização adiciona componente

Na Figura 5.3, o pacote “methods” contem alguns pacotes utilizados para a organização dos métodos, como analysisOperators, areaOperators, entre outros. Deseja-se incluir o método de segmentação por Watershed, como o mesmo se encaixa no grupo de métodos que trabalham utilizando área da imagem, ele deve ser colocado no pacote areaOperators. É importante lembrar que o método funciona independentemente de estar nesse pacote, porém, como forma de organização e facilidade, o grupo adotou essa padronização.

No que tange a documentação do sistema, a qual não era realizada pelo grupo, passou a ser um item importante para garantir as informações. A documentação ocorreu como propõe a MetImage, na forma do Javadoc e dos relatórios. A equipe de desenvolvimento utilizou a ferramenta de documentação DokuWiki[12] como forma de agilizar o processo de documentação, vindo com isso conter o conteúdo dos relatórios gerados pela MetImage. A vantagem de usar essa ferramenta está no fato que a mesma permite que os documentos sejam editados coletivamente apenas através da utilização de um navegador web.

Os testes que já eram feitos foram incrementados com os testes de aceitação e os mesmos também foram padronizados. Para realização dos testes de unidade utilizou-se a ferramenta Junit. O principal motivo para utilizar essa ferramenta é pelo fato de ser livre e permitir a automatização dos testes escritos em Java. Os testes de aceitação foram realizados pelo usuário interno, onde o software foi usado com o testador acompanhando e registrando erros e problemas de uso. Os itens observados foram a facilidade de compreensão do sistema pelo usuário e a adaptabilidade do software ao modo de trabalho do usuário.

Inicialmente a integração do sistema era de responsabilidade de um membro da equipe, o qual recebia a nova funcionalidade implementada e incluía no sistema, ficando a seu cargo disponibilizar a nova versão em uma máquina servidor do grupo, depois de feito isso os demais integrantes eram avisados da nova versão por e-mail. Mas tendo em vista que esse tipo de trabalho acarretava em sobre carga em um único integrante, a equipe de desenvolvimento em conjunto com o gerente de projeto passaram a utilizar uma ferramenta para fazer o controle de versões. A

[12] Desenvolvida por Andreas Gohr. (http://wiki.splitbrain.org/wiki:dokuwiki).

ferramenta escolhida foi SVN[13] a qual permite que várias pessoas editem os arquivos ao mesmo tempo, mantendo um histórico das versões anteriores.

O gerenciamento do projeto foi realizado através das reuniões semanais, nas quais os desenvolvedores apresentavam o que tinha sido implementado, as dúvidas, e a definição de novas tarefas.

5.2.3 Etapa Transição

Tendo em vista que a aplicação da metodologia proposta ocorreu em um projeto de desenvolvimento de software já iniciado anteriormente, e que o mesmo encontra-se em processo de desenvolvimento, a etapa de transição que é proposta pela MetImage não pode ser realizada.

5.2.4 Pontos Relevantes sobre MetImage no Escopo do Projeto

Com a aplicação da MetImage no contexto de um grupo de pesquisa, mesmo o projeto já tendo sido iniciado e estar em andamento, pode-se constatar algumas vantagens em fazer uso dessa metodologia.

As vantagens de se usar a metodologia proposta estão centradas no fato da mesma possuir a etapa necessária para o entendimento dos métodos matemáticos necessários para implementação da solução do problema. Essa etapa é de extrema importância tendo em vista que o processamento e análise de imagens é uma técnica extremamente dependente do problema que se quer resolver e muitas das técnicas de processamento e análise utilizadas para resolver os problemas estão baseadas em métodos matemáticos que permitem descrever quantitativamente imagens das mais diversas origens.

---

[13] Copyright © 2002, 2003, 2004, 2005, 2006 Ben Collins-SussmanBrian W. FitzpatrickC. Michael Pilato.

Outra vantagem está na documentação, sabendo-se que o excesso de documentação é igual a nenhuma, a documentação gerada pelo grupo ao utilizar a metodologia auxilia a otimizar a comunicação entre integrantes de áreas diferentes. Essa otimização ocorre através do relatório de implementação, o qual além de conter o resumo da funcionalidade implementada e da sua aplicação, possui o diagrama de classe da funcionalidade e uma explicação sobre os métodos matemáticos utilizados para a implementação. Além de também situar os novos membros da equipe sobre o que foi e como foi desenvolvido o sistema.

Outro ponto de bastante importância é a etapa de integração de novos membros na equipe. Essa etapa é de extrema importância, pois permite que os novos membros da equipe sejam integrados no grupo. Essa integração ocorre através dos seminários de integração e através da documentação.

5.3 Comparação do XP e da Metodologia MetImage

Serão examinados simultaneamente os recursos da metodologia eXtreme Programming e da metodologia MetImage para determinar as semelhanças e/ou relações entre ambas.

A comparação pode ser visualizada na Tabela 5.1.

Tabela 5.1 - Comparação das metodologias

RECURSO	XP	METIMAGE
Papéis	• Programador • Cliente • Testador • Tracker • Coach • Analista • Gerente	• Gerente • Desenvolvedor • Testador • Usuário externo • Usuário interno
Cliente	• Deve fazer parte da equipe	• Deve fazer parte da equipe
Documentação	• Cartões de história (temporário) • Direto no código	• Relatório de implementação • Relatório de testes • Diagrama de classe • Relatório do projeto • Direto no código
Programação	• Em pares	• Individual
Refactoring	• Utiliza	• Utiliza
Testes	• Teste de unidade	• Teste de unidade • Teste de aceitação
Ciclo de Vida	• Iterativo incremental	• Iterativo incremental
Diagramas	• Não utiliza	• Diagrama de classe
Número de programadores	• 2 a 10 programadores (no máximo)	• Não tem limite
Integração novos membros	• Não possui	• Possui, sendo realizada através dos seminários de integração
Definição da arquitetura	• Na fase de exploração	• Na fase de elaboração

6 CONCLUSÃO

A dependência e demanda crescentes da sociedade em relação à Informática e, em particular, ao software, tem ressaltado uma série de problemas relacionados à forma de desenvolvimento de software, como alto custo, alta complexidade, entre outros.

A maior parte das metodologias existentes é baseada em premissas bastante tradicionais da área: uma divisão discreta das atividades do processo de software, focando em gerenciamento, medidas e registros destas atividades.

No âmbito de processamento e análise de imagens, apesar de ainda não haver referências específicas sobre o uso de uma metodologia nessa área, notou-se que fazer uso de uma metodologia que direcione a equipe é de vital importância, isso ocorre tendo em vista que a equipe necessária para esse tipo de desenvolvimento necessita ser multidisciplinar.

Esse trabalho apresentou a aplicação da MetImage na ferramenta Arthemis, o propósito da sua aplicação foi direcionar as atividades dentro da equipe de desenvolvimento, otimizar a comunicação entre as áreas diferentes, facilitar a integração de novos membros na equipe e agilizar/organizar o processo de desenvolvimento.

Mesmo a metodologia estar sendo aplicada e necessitar de validação, os principais resultados obtidos até o presente momento dizem respeito à importância da etapa de aprendizagem sobre os métodos matemáticos necessários para a implementação das funcionalidades, a utilização de uma padronização no desenvolvimento do código e da documentação e a documentação gerada pela equipe que serviu de apoio para o entendimento entre especialistas das diferentes áreas integrantes no grupo.

A principal limitação encontrada ao realizar esse trabalho diz respeito à aplicação das metodologias. Tanto a metodologia eXtreme Programming quanto a MetImage foram aplicadas em um projeto que estava em andamento. Isso teve como resultado a não aplicação de algumas etapas da metodologia proposta. Essas

etapas são a de Iniciação e a de Transição. Essa última tendo em vista que o projeto não havia sido finalizado no período da aplicação da MetImage.

Na aplicação da metodologia eXtreme Programming além do projeto estar em andamento houve resistência por parte da equipe em utilizar uma metodologia que não gerava nenhuma documentação e realizava reuniões diárias.

A principal contribuição desse trabalho para sua área de abrangência está no desenvolvimento de uma metodologia específica para a construção de software em processamento e análise de imagens, a qual tenta suprir as barreiras das metodologias existentes, que estão relacionadas ao excesso de recursos, burocracia, controle exagerado e por parte de algumas (XP, por exemplo) falta de documentação.

Por outro lado, para o contexto do grupo de pesquisa onde foi realizado o estudo de caso, a principal contribuição da aplicação da metodologia MetImage foi a organização do trabalho da equipe como um todo, uma vez que antes da sua aplicação não existia a definição de procedimentos quanto a padronização de código, documentação sobre os sistemas desenvolvidos, controle de versão, necessidade de uma etapa específica para aprendizado sobre métodos matemáticos e para integrar novos membros na equipe.

Sendo essa última de fundamental importância, tendo em vista que o grupo recebe novos integrantes todos os semestres, e uma forma de integrar esses novos membros são através dos seminários de integração, o qual tem a finalidade de apresentar projeto e situar os novos membros dentro da equipe.

6.1 Trabalhos Futuros

Este trabalho pode ser estendido de várias formas visando aumentar a contribuição para a área de processamento e análise de imagens, que deseja se beneficiar do uso de metodologias específicas para esse tipo de desenvolvimento.

Alguns trabalhos sugeridos são:

- Aplicar a metodologia proposta em um projeto de desenvolvimento de software que trabalhe com visão computacional no contexto de uma incubadora tecnológica;

- Incluir o diagrama de Gantt para ilustrar o avanço das diferentes etapas do projeto e mostrar quem é responsável por cada atividade e para quando está programado o início e o término da atividade;
- Aplicar técnicas de Interação Humano-Computador (IHC), como métodos observacionais e técnicas de questionamento, em conjunto com os testes de aceitação como forma de avaliar a funcionalidade do sistema, o efeito da interface junto ao usuário e identificar os problemas.

REFERÊNCIAS

ALBUQUERQUE, Márcio Portes de; ALBUQUERQUE, Marcelo Portes de ; CANER, Eugenio Suarez ; GESUALDI, Aline da Rocha . **Análise de imagens e visão computacional**. In: ANAIS DA V ESCOLA DO CBPF. Rio de Janeiro, v. 1, p. 145-176, 2005.

BECK, Kent. **Extreme Programming Explained:** Embrace Change. 2.ed. Addison Wesley, 2000.

BELCHIOR, A. D. **Um modelo fuzzy para avaliação da qualidade de software**. 1997. 130f. Tese (Doutorado em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1997.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. Editora Campus, 2003.

BLOIS, A. P. B. **Uma proposta de projeto arquitetural baseado em componentes no contexto de engenharia de domínio**. 2004. 134f. Tese (Doutorado em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2004.

BROWN, Alan. **Large-scale component-based development**. Prentice Hall Books. 2000.

BOEHM, Barry. Managing software productivity and reuse. **IEEE Computer**, v.32 (9), p.111-113, 1999.

BOEGH, J. et al. **A practitioners guide to evaluation of software.** In: SOFTWARE ENGINEERING STANDARDS SYMPOSIUM. Brighton, UK. Qualidade de software: Teoria e prática, Prentice Hall, 1993.

BONA, Cristina. Avaliação de Processos de Software: Um Estudo de Caso em XP e Iconix. 2002. 122f. Dissertação (Mestrado em Engenharia da Produção) – Universidade Federal de Santa Catarina, Florianópolis, 2002.

BUSCHMANN, Frank; MEUNIER, Regine; H. R. P. S. M. S. **Pattern - Oriented software architecture: A system of patterns.** v1. John Wiley and Sons Ltd. 1996.

CAGNIN, M. I. **Parfait: Uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks**. 2005. 241f. Tese (Doutorado em Ciências da Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos, 2005.

CHEESMAN, J.; DANIELS, J. **UML components: a simple process for specifying component-based software**. Addison Wesley, 2001.

CLEMENTS, Paul. C.; NORTHROP, Linda. M. **Software Architecture: An Executive Overview.** Technical Report CMU/SEI-96-TR-003, Software Engineering Institute, Carnegie Mellon University, 1996.

COELHO, C. C. **Maps: um modelo de adaptação de processo de software**. 2003. 178f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Pernambuco, Recife, 2003.

CONALLEN, Jim. **Building web applications with UML**. Addison Wesley, 2000.

DEMARCO, Tom. **Software development: State of the art vs. state of the practice** In: WHY DOES SOFTWARE COST SO MUCH? New York: Dorset House, 1st. ed., 1995.

D'SOUZA, Desmond.; WILLS, Alan. **Objects components and frameworks with UML: The Catalysis Approach**. 1. ed. Massachusetts, Addison Wesley, 1999.

ECKEL, Bruce. **Thinking in Java**. 3. ed. New Jersey: Prentice Hall, 2002.

ERICH GAMMA, RICHARD HELM, R. J.; VLISSIDES, J. **Design patterns - elements of reusable object-oriented software**. Bookman, 2000.

ENGELS, Gregor.; GROENEWEGEN, Luuk. Object-oriented modeling: A roadmap. **Communications of the ACM**, 43(5):103-116, 2000.

FAYAD, Mohamed. E.; SCHMIDT, D. C.; JOHNSON, Ralph. E. **Building application frameworks: object-oriented foundations of framework design**. New York: John Wiley and Sons, 1999.

FILHO, G. da C.; PENTEADO, R.; SILVA, J. A.; BRAGA, R.T.V. **Padrões e Métodos Ágeis: agilidade no processo de desenvolvimento de software**. In: 5th LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING - SugarLoafPloP 2005. p. 145-157. 2005.

FUGGETA, Alfonso. **Software Process: a Roadmap**. In: 22nd INTERNATIONAL CONFERENCE ON THE FUTURE OF SOFTWARE ENGINEERING. Limerick, Ireland: ACM, p.25-34. 2000.

GIMENES, I. M. DE S.; HUZITA, E. H. M. Desenvolvimento baseado em componentes: conceitos e técnicas. Rio de Janeiro: Editora Ciência Moderna. 2005.

KELLNER, M. I. **Software process modeling experience**. In: PROCEEDINGS OF ICSE, 11., Pittsburgh. IEEECSP. p. 400-401, 1989.

LARMAN, C. **Applying UML and Patterns: an introdution to object-oriented analysis and desing and iterative development**. 3.ed. Prendice Hall, 2004.

LINDVALL, M.; RUS, I. Process diversity in software development. **IEEE Software**, v. 17, n. 4, p. 14-18, July/August, 2000.

MACHADO, L.F.C. **Modelo para definição de processos de software na Estação TABA**. 2000. 150f. Dissertação (Mestrado em Ciências em Engenharia de Sistemas e Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2000.

MACIASZEK, L. A. **Requirements analysis and system design: developing information systems with UML**. Addison Wesley. 2000.

MARCUS VÖLTER, A. S.; WOLFF, E. **Server Component Patterns: Component Infrastructures Illustrated with EJB**. John Wiley & Sons. 2002.

MCCLURE, C. **Software reuse techniques: adding reuse to the system development process**. 1.ed. New Jersey: Prentice Hall, 1997.

METSKER, S. J. **Design patterns java workbook.** Addison Wesley. 2002.

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA / SECRETARIA DE POLÍTICA DE INFORMÁTICA (MCT / SEPIN). **Qualidade e produtividade no setor de software**. 1999.

PRESSMAN, R. **Engenharia de software: uma abordagem prática**. 5.ed. Rio de Janeiro: McGraw-Hill, 2002.

REIS, C. **Caracterização de um modelo de processo para projetos de software livre.** 2001. 46f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos, 2001.

ROCHA, A. R. C; MALDONADO, J. C; WEBER, K. C. **Qualidade de Software - Teoria e Prática**. Prentice Hall, 2001.

ROSSI, A. C. **Representação do componente de software na FarcSoft: ferramenta de apoio à reutilização de componentes de software**. 2004. 253f. Dissertação (Mestrado em Engenharia) – Universidade de São Paulo, São Paulo, 2004.

SALVIANO, C.F. **Introdução a software patterns**. In: XI SBES, Fortaleza, Ceará-Brasil, p. 22. 1997.

SAMETINGER, J. **Software engineering with reusable components**. Springer, 1997.

SHAW, M. **The coming-of-age of software architecture research**. In: PROCEEDINGS ON THE 23rd INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE 2001. Keynote address. May.Toronto, Canada. 2001.

SMITH, John. **A Comparison of RUP and XP**. Rational Software White Paper. 2001. Disponível em http://www.rational.com. Acesso em: 3 out. 2005.

SOMMERVILLE, Ian. **Software Engineering**. 6.ed. New York: Addison Wesley, 2001.

SOUZA, G. T. de; PIRES, C. G.; MARINHO, F. G.; BELCHIOR, A. D. **Aplicação de metapadrões e padrões em desenvolvimento de software para sistemas de informação**. In: 5th LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING - SugarLoafPloP 2005. p. 167-179. 2005.

TIBERTI, A. J. **Desenvolvimento de software de apoio ao projeto de arranjo físico de fábrica baseado em um framework orientado a objeto**. 2003. 195f. Tese (Doutorado em Engenharia Mecânica) - Universidade de São Paulo, São Carlos, 2003.

WELFER, D. Padrões de projeto no desenvolvimento de sistemas de processamento de Imagens. 2005. 81f. Dissertação (Mestrado em Engenharia da Produção) – Universidade Federal de Santa Maria, Santa Maria, 2005.

WILLIAMS, L.; COCKBURN, Alistair. Agile software development: it's about feedback and change. **IEEE Computer**, v.36, n.6, p.39-43, 2003.

APÊNDICE A – Template Relatório do Projeto

Relatório do Projeto

Versão:
Data:

1- Introdução
- 1.1 Apresentação do projeto
- 1.2 Objetivos
- 1.3 Metas
- 1.4 Usuários

2 – Organização do Projeto < Descreve o modo como a equipe é organizada, as pessoas envolvidas e seus papéis >
- 2.1 Papéis e Responsabilidades
- 2.2 Equipe < Descreve a equipe do projeto e seus respectivos papéis >

3 – Requisitos de Software e Hardware

4– Programação do Projeto < Descreve as dependências das atividades, quando houver, e o tempo estimado para realizar cada atividade >

5 – Procedimentos, Métodos e Padrões <Descreva todos os procedimentos, métodos e padrões adotados para desenvolver e/ou manter o software >

Procedimento/método/padrão	Descrição
Padrão de codificação	Padrão utilizado para codificar *software*.

6 – Ferramentas

Ferramenta	Descrição

7 – Arquitetura do Sistema

7.1 Descrição da Arquitetura

7.1 Diagrama da Arquitetura

8 - Controle de Versões

9 – Cronograma < Cronograma de releases e iterações >

APÊNDICE B – Template Relatório de Implementação

Relatório de Implementação

Desenvolvedor:
Data:

1- Nome da Funcionalidade

_____

2- Objetivo da Funcionalidade

_____
_____
_____

3- Descrição da Funcionalidade

_____
_____
_____

4- Diagrama de Classe

5- Métodos Matemáticos Utilizados

6- Observações

_____
_____
_____

APÊNDICE C – Template Relatório de Teste de Unidade

Relatório de Teste de Unidade

Item de Teste:			Data:
Desenvolvedor:		Testador:	
Objetivo do Teste:			
Dependência/Pré-requisito:			
Casos de Teste			
Entrada	Resultado Esperado	Resultado Encontrado	Situação