Manoel Pereira Junior

CONCEPÇÃO DE UM PROCESSO DE DESENVOLVIMENTO ESPECÍFICO PARA SOFTWARE CIENTÍFICO

Manoel Pereira Junior

CONCEPÇÃO DE UM PROCESSO DE DESENVOLVIMENTO ESPECÍFICO PARA SOFTWARE CIENTÍFICO

Dissertação apresentada ao Curso de Mestrado em Modelagem Matemática e Computacional (MMC) do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Modelagem Matemática e Computacional.

Área de pesquisa: Sistemas Inteligentes.

Orientador: Prof. Dr. Gray Farias Moita

Belo Horizonte
Centro Federal de Educação Tecnológica de Minas Gerais
Diretoria de Pesquisa e Pós-Graduação
2007

Folha de aprovação. Esta folha será fornecida pelo Programa de Pós-Graduação e deverá substituir esta página.

Dedico este trabalho aos meus pais Manoel e Eliana, e também

à minha irmã Camila, que sempre me incentivam

a seguir em frente e a buscar cada

vez mais conhecimento.

À minha amada e futura esposa Amanda, que compreendeu

meus momentos de ausência e angústia

e nunca me deixou desistir.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Gray Farias Moita, por mostrar os caminhos do conhecimento e do mundo, pelo incentivo durante todo o projeto e pela paciência nos momentos de angústia.

Aos colegas e amigos da Fundação Educacional de Oliveira e do Centro Federal de Educação Tecnológica de Bambuí pelo apoio nos momentos de dificuldade e ausência.

Aos professores e colegas do Grupo de Pesquisa em Sistemas Inteligentes – GPSI, do CEFET-MG, pelas críticas e sugestões durante a prática dos seminários do grupo.

À todos aqueles que contribuíram de alguma forma com este trabalho.

"Sábio é aquele que conhece os limites da própria ignorância." (Sócrates)

Resumo

A geração de sistemas computacionais confiáveis e de alta qualidade requer, fundamentalmente, a adoção de um processo de desenvolvimento adequado. Dentro deste tema, uma questão que tem merecido a atenção dos pesquisadores é o estudo do processo de desenvolvimento de software científico de natureza acadêmica. Uma das motivações para tal preocupação é a existência de vários modelos de processos para a construção de software, porém, dedicados a software convencionais. Outros tipos de software, como os criados por cientistas para serem utilizados em suas pesquisas, possuem particularidades e preocupações diferentes dos sistemas convencionais e comerciais, tais como a alta rotatividade da equipe de desenvolvimento, o papel do cliente e a descoberta dos requisitos ao longo do desenvolvimento do software, o que pode tornar os processos existentes inadequados para o desenvolvimento de software científico. Este trabalho tem como objetivo a definição de um processo de desenvolvimento de software científico de natureza acadêmica, denominado PESC (Processo de desenvolvimento Específico para Software Científico), com a intenção de conferir qualidade, no sentido mais amplo do termo, ao software científico. Tal processo foi concebido com base nas diretrizes já identificadas em uma etapa anterior da pesquisa e com o apoio da literatura. Espera-se com este trabalho, definir um processo inicial para o PESC.

PALAVRAS-CHAVE: Engenharia de Software; Processo de Desenvolvimento de Software; Software Científico.

Abstract

The generation of trustworthy and high quality computational systems, requires, basically, the adoption of an adequate development process. Within this subject, a question that it has deserved the attention of the researchers is the study of the process of development of scientific software of academic nature. One of the motivations for such concern is the existence of different types of processes for the construction of software, however, dedicated software conventional. Other kind of software, as those created by scientists to be used in their research, might have particularitities and possibly different concerns from the conventional and commercial systems. Hence, the existing processes for the development of scientific software might become inadequate. This work has the objective of defining a development process for scientific software of academic nature, here called PESC (Process of Specific Development for Scientific Software), with the intention to confer quality to these software. Such process was conceived on the basis of the directions indicated in the earlier phase of the work and with the support of the literature. It is expected, with this research, to define the PESC initial process.

KEY WORDS: Software Engineering; Software Development Process; Scientific Software.

LISTA DE ABREVIATURAS E SIGLAS

CASE Computer Aided Software Engineering

CEFET-MG Centro Federal de Educação Tecnológica de Minas Gerais

GPSI Grupo de Pesquisas em Sistemas Inteligentes

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos

LSI Laboratório de Sistemas Inteligentes

OO Orientação a Objetos

OOP Object-Oriented Programming

PESC Processo de desenvolvimento Específico para Software Científico

POO Programação Orientada a Objetos

PSP Personal Software Process

PU Processo Unificado

RAD Rapid Application Development

RUP Rational Unified Process

UML Unified Modeling Language

XP Extreme Programming

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

Com a crescente evolução do seu poder de armazenamento e de processamento e uma considerável redução de custos, os computadores estão cada vez mais presentes na sociedade atual, que se vê cada vez mais dependente destas máquinas e de seus programas, também conhecidos como software. Estes, por sua vez, têm apresentado um significativo aumento na complexidade interna, fato este que favorece a maior incidência de erros e, conseqüentemente, queda na sua qualidade. Segundo Silva *et. al.* (2003), a qualidade do software é menos adequada do que deveria. A qualidade dos software é freqüentemente suspeita.

Técnicas de Engenharia de Software são empregadas nos casos em que se deseja obter a garantia da qualidade do software que será desenvolvido. Estas técnicas - conhecidas como Processos de Desenvolvimento de Software - quando bem empregadas, possibilitam um desenvolvimento de software de alta confiabilidade e qualidade.

Um processo de desenvolvimento de software pode ser definido como uma coleção de fatores necessários para a construção de software de alta qualidade. Segundo Ambler (1998), um processo, ainda, pode ser definido como uma série de ações na qual uma ou mais entradas são utilizadas para produzir uma ou mais saídas.

Os termos qualidade e confiabilidade têm merecido uma atenção especial dos pesquisadores quando se foca o processo de desenvolvimento de software moderno. Existem vários processos de desenvolvimento gerados para nortear a elaboração de software comerciais e garantir que tanto a confiabilidade quanto a qualidade sejam asseguradas. No entanto, nota-se claramente uma carência destes processos, quando o que se pretende desenvolver é um software científico ou acadêmico, que apresenta particularidades distintas do software comercial.

Uma das particularidades que o software científico apresenta em relação ao software comercial é o papel do cliente. No desenvolvimento de um software comercial, o cliente tem o seu papel bem definido. Ele é a peça principal da engenharia de requisitos e, ao final do desenvolvimento, deve certificar-se que o acertado foi realmente implementado. No desenvolvimento de um software científico não existe o papel do cliente. Este tipo de software geralmente é construído por um pesquisador, com a finalidade de validar ou apoiar sua própria pesquia. Sendo assim, os pontos de validação existentes no desenvolvimento de um software comercial inexistem no desenvolvimento de um software científico.

Outra particularidade peculiar do desenvolvimento do software científico é a identificação dos requisitos ao longo do seu desenvolvimento. Quando se desenvolve um software comercial, geralmente já se tem a noção, mesmo que inexata, do sistema como um todo, ou seja, de todas as suas funcionalidades. Já no software científico os requisitos podem mudar, por exemplo, pela evolução da pesquisa. Sendo assim, o pesquisador não tem o conhecimento dos detalhes do software científico como um todo, diferentemente do software comercial.

Uma outra particularidade ainda pode ser observada. A alta rotação dos pesquisadores em uma pesquisa. Geralmente um software científico é desenvolvido para validar uma determinada parte de uma pesquisa e, ao término da pesquisa, o desenvolvedor se desvincula da pesquisa, como acontece com os alunos de mestrado. No entanto, os próximos alunos freqüentemente continuam o desenvolvimento do software. Já no software comercial, existe uma equipe de desenvolvimento pré-definida e não se altera até o término do projeto. Além disso, no desenvolvimento de um software comercial existem pessoas com funções distintas, como os analistas de sistemas, os administradores da base de dados, engenheiros de teste, e os programadores. No desenvolvimento de um software científico o pesquisador desempenha todas estas atividades.

A utilização de um processo formal de desenvolvimento de software é crucial em um desenvolvimento de sucesso, principalmente quando o foco do desenvolvimento é um software de natureza acadêmica, que geralmente é desenvolvido para fundamentar pesquisas científicas. Um processo é importante porque fornece controle e estabilidade para a atividade de desenvolvimento de software, que pode se tornar caótica, se deixada de lado.

Segundo Cordeiro (2000), os processos usados para desenvolver um projeto de software têm a maior importância na qualidade do software produzido e na produtividade alcançada pelo projeto. Por conseqüência, existe uma necessidade de melhorar os processos usados em uma organização para desenvolver projetos de software.

Procurando atender à demanda por um processo para desenvolvimento de software científico, iniciou-se um estudo para a concepção de tal processo, denominado PESC (Processo de desenvolvimento Específico para Software Científico). Em um primeiro momento da pesquisa foram identificadas as particularidades do software científico em relação ao software convencional e foram geradas diretrizes para a concepção de um processo inicial para o PESC, com base em uma pesquisa realizada com a comunidade que desenvolve software científicos (Purri, 2006).

Portanto, o presente trabalho se focou no estudo das diretrizes geradas anteriormente e na proposição de um processo inicial para o PESC (*Processo de desenvolvimento Específico para Software Científico)*, com o apoio da literatura.

Espera-se que o PESC possa contribuir com os pesquisadores que desenvolvem software científicos, a fim de apoiar suas pesquisas, permitindo que o produto a ser gerado seja bem documentado, aprimorando assim o seu desenvolvimento cotidiano.

1.1 Motivação

A utilização de software dos mais variados tipos, em larga escala, hoje é um fato conhecido. Além disto, existe uma tendência de aumento desta utilização. Sendo assim, os sistemas computacionais devem naturalmente apresentar cada vez mais sofisticação. No entanto, estes fatores têm influência direta na qualidade e confiabilidade do software, já que a sua complexidade tende naturalmente a aumentar.

A comunidade científica, por sua vez, tem utilizado cada vez mais os computadores e software para o desenvolvimento de suas pesquisas. Tais software têm que apresentar uma altíssima confiabilidade e qualidade, pois, em geral, servirão para validar ou comprovar pesquisas científicas. Porém, segundo Purri (2006), os desenvolvedores de software científico possuem grande carência acerca de um processo de desenvolvimento específico para a concepção de seus sistemas computacionais.

Além disso, nota-se que os desenvolvedores de software científicos não necessariamente conhecem as técnicas clássicas de engenharia de software, como identificado por Purri (2006) em seu trabalho de mestrado. Cada um destes desenvolvedores utiliza técnicas próprias para o controle do seu desenvolvimento, tornando-o um tanto quanto pessoal. Este fato ocasiona uma série de desvantagens, tais como:

- Falta de organização e de reutilização;
- Falta de continuidade nas pesquisas e no desenvolvimento;
- Falta de otimização;
- Retrabalho;
- Documentação Inexistente.

A motivação deste trabalho é exatamente a definição de um processo inicial para o PESC que, como processo de desenvolvimento de software científico, irá cadenciar o seu desenvolvimento, impedindo que este se torne caótico a

ponto de tornar o projeto inviável. Além disso, o processo deve conferir a estes software maior confiabilidade e qualidade, tanto na sua estrutura interna (legibilidade do código-fonte, reusabilidade de componentes, dentre outros) quanto na documentação que faz parte do software (documentos de análise, projeto, manuais do usuário e demais documentos gerados).

1.2 Caracterização do Problema de Pesquisa

Os desenvolvedores de software comerciais dispõem hoje de uma série de processos de desenvolvimento específicos para este fim, como o Processo Unificado e a Extreme Programming (XP). No entanto, nota-se claramente uma carência de um processo de desenvolvimento próprio para fins científicos que, segundo Purri (2006), possuem particularidades e preocupações aparentemente diferentes dos sistemas convencionais e comerciais.

O grande problema tratado nesta pesquisa é exatamente esta falta de um processo de controle sobre a atividade de desenvolvimento de software científicos. Como já foi dito anteriomente, o processo praticado atualmente pelos pesquisadores que desenvolvem software para fundamentar suas pesquisas, é um tanto quanto pessoal (Purri, 2006). Este fato gera uma grande dificuldade na continuação, reuso e organização do sistema. Para atacar este problema, o presente trabalho apresenta a definição de um Processo de desenvolvimento Específico para Software Científico, denominado PESC.

1.3 Estrutura do Trabalho

Capítulo 2 – Fundamentação Teórica

Neste capítulo são apresentados os fundamentos da pesquisa, com ênfase nos processos de desenvolvimento de software, que são a base desta pesquisa, além de tópicos como Engenharia de Software e modelos de desenvolvimento de software.

Capítulo 3 – Pressupostos para a Construção do Processo Inicial

Este capítulo apresenta as diretrizes que deram origem ao processo inicial do PESC. Caracteriza os passos anteriores da pesquisa e o que foi desenvolvido neste trabalho.

Capítulo 4 – Processo Proposto

O processo proposto, com base nas diretrizes indicadas, são mostrados neste capítulo. O processo inicial gerado, o seu ciclo de desenvolvimento e seus artefatos são detalhados e explicados.

Capítulo 5 – Análise do Processo Proposto

Este capítulo apresenta a análise do processo proposto. Neste capítulo, os artefatos gerados são apresentados e analisados, segundo os pilares indicados na fase anterior da pesquisa, que foi caracterizada no capítulo 3.

Capítulo 6 – Conclusão

Neste último capítulo, são apresentadas as conclusões da pesquisa, bem como algumas considerações finais. São explicitadas ainda algumas propostas de desenvolvimento futuro.

2. Fundamentação Teórica

A grande área do conhecimento à qual este trabalho se relaciona é a Engenharia de Software. Um dos ramos desta disciplina que tem merecido bastante atenção da comunidade científica é o de qualidade de software. Com base neste contexto, dentro deste capítulo são apresentados os conceitos básicos de Engenharia de Software, dos modelos e processos de desenvolvimento de software e demais conceitos relevantes para propiciar um adequado entendimento deste texto.

2.1 Engenharia de Software

Segundo Sommerville (2003), a Engenharia de Software pode ser definida como:

> "uma disciplina de engenharia que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação."

Pressman (2006) apresenta ainda uma outra definição para engenharia de software:

> "tecnologia que abrange um processo, um conjunto de métodos e ferramentas para construção de software de computador. É a criação e utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais."

A engenharia de software, de acordo com as definições acima, pode ser descrita como um arcabouço de técnicas e metodologias para a documentação e conseqüente controle do processo de desenvolvimento de software que, quando utilizada de forma correta, propicia um software de qualidade assegurada.

O software, por sua vez, pode ser definido como o produto com que a engenharia de software se preocupa. Em outras palavras, é o código executável por um computador, produzido por engenheiros de software, que reflete as implementações das necessidades de uma determinada situação. Tal produto é hoje é fator crucial no quesito competitividade empresarial. O software pode influenciar, por exemplo, na tomada de decisões nos negócios. Estes fatos tornam inviável a existência de uma grande empresa que queira ser competitiva no mercado sem a sua utilização. O

software hoje deve ser eficiente o suficiente para entregar a informação necessária, na hora correta, à pessoa interessada.

Mas, tão importante quanto a eficiência de um software, é a sua qualidade. Um software que seja bastante eficiente, mas que não gere os resultados satisfatórios ou que não implemente as funcionalidades desejadas, obviamente não será bem sucedido.

Segundo Paula (2001), a qualidade de um software é proporcional ao seu grau de conformidade com os respectivos requisitos. Sendo assim, quanto mais preciso for um software, no sentido de atingir as necessidades identificadas, mais qualidade ele terá. Cabe aqui ressaltar que existem requisitos funcionais e requisitos não funcionais. Segundo Sommerville (2003), os requisitos funcionais indicam as funções que o software deverá implementar. Os requisitos não-funcionais englobam questões como performance, usabilidade e adaptabilidade. Nota-se, ainda, que quanto maior a qualidade de um software, maior será a sua confiabilidade e conseqüente utilização e aceitação. A qualidade então é o centro da engenharia de software.

A qualidade de um software é conseguida aplicando-se um conjunto de métodos e ferramentas. Este conjunto de métodos e ferramentas é gerenciado por um processo. Têm-se, então, segundo Pressman (2006), três fundamentos da Engenharia de Software: Ferramentas, Métodos e Processos. Os métodos proporcionam todos os detalhes de como fazer para construir um software. Neste fundamento, são estabelecidos os critérios e as formas de trabalho da equipe de desenvolvimento. Os métodos, quando aplicados de forma correta e coerente, permitem o gerenciamento ordenado do ciclo de desenvolvmento do software, fator este que é crucial para a garantia da sua qualidade.

As ferramentas dão suporte automatizado aos métodos. Quando as ferramentas são integradas é estabelecido um sistema de suporte ao desenvolvimento de software chamado CASE (Computer Aided Software Engineering).

Segundo Pressman (2006), os processos são o elo de ligação entre os métodos e as ferramentas. Um processo sistemático deve ser aplicado, em qualquer ramo do conhecimento, sempre que houver a necessidade de qualidade. Deve-se levar em conta que em qualquer processo existem pessoas envolvidas. A qualidade depende então, também, do empenho destas na aplicação do processo em que estão inseridas. Estes conceitos dão suporte à Figura 2.1.

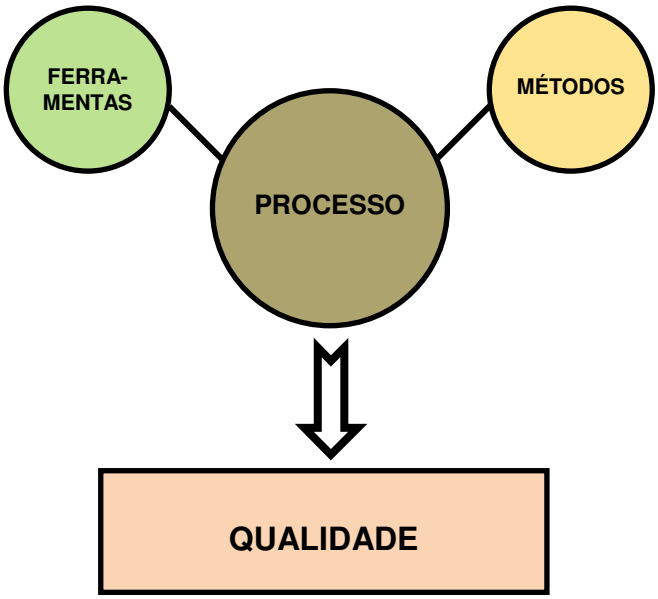

FIGURA 2.1 – Obtenção da qualidade por meio de um processo

Existe uma abordagem genérica para a aplicação de um processo de engenharia de software. Tal abordagem é dividida em 3 fases genéricas (Pressman, 2002; Sommerville, 2003). A primeira fase, que é a de definição, concentra-se em definir o que o software deverá fazer e como funcionará. Já na segunda fase, a de desenvolvimento, acontece a codificação propriamente dita, além da realização de testes. A terceira e última fase, a de manutenção, focaliza as modificações que podem ocorrer no software ao longo de sua vida, como correção de erros, adaptações necessárias e melhoramentos solicitados pelos clientes. Este ciclo de melhoramentos se prolonga até que todas as necessidades de aplicação do software sejam satisfeitas. A Figura 2.2 apresenta as três fases genéricas do desenvolvimento de um determinado software.

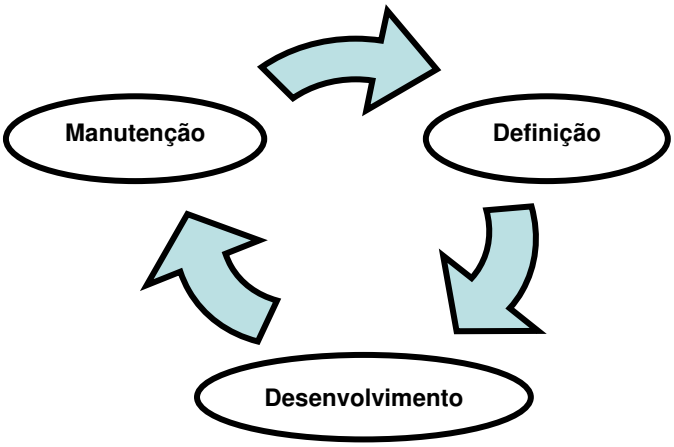

FIGURA 2.2 – Três fases genéricas da Engenharia de Software.

Vale salientar que o “software se deteriora, diferentemente do hardware que se desgasta” (Pressman, 2002). O hardware, com o passar do tempo sofre com as influências do meio onde ele está inserido, como poeira e humidade. Este fato dá base para a criação da chamada “curva da banheira”, que caracteriza as falhas do hardware. A curva da banheira pode ser vista na Figura 2.3.

FIGURA 2.3 – Curva da banheira (Fonte: Adaptado de PRESSMAN, 2006)

A curva de falhas do software deveria apresentar uma alta taxa de erros no início de sua vida, que normalmente são corrigidos como no caso do hardware e depois manter um nível de falhas altamente baixo e constante (curva ideal), já que o meio não influencia na sua vida. No entanto o software necessita constantemente de manutenção. Essas alterações geralmente trazem falhas agregadas que novamente elevam a taxa de falhas, causando então um recomeço do processo de correção de falhas (curva real). A Figura 2.4 mostra as curvas real e ideal do software.

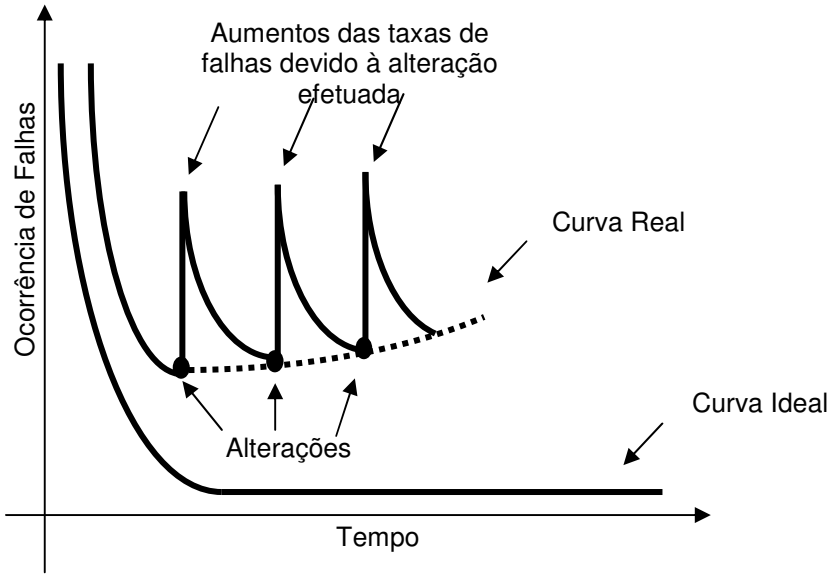

FIGURA 2.4 – Curvas do software (Fonte: Adaptado de PRESSMAN, 2006)

Exatamente para evitar estas possíveis falhas que são acrescentadas nas alterações de um sistema, um bom processo deve ser empregado. Este processo deve ser capaz de prever e tratar as alterações necessárias no software, de modo que a qualidade não seja comprometida.

O software científico, assim como o software comercial, também tem suas falhas representadas pela curva mostrada na Figura 2.4, onde a cada nova alteração são inseridas novos problemas.

A seção a seguir apresenta alguns modelos de processos de desenvolvimento de software tradicionais, que formaram a base para a concepção do ciclo de desenvolvimento e dos artefatos que compõem o processo inicial do PESC (Processo de desenvolvimento Específico para Software Científico).

2.2 Modelos de Processo de Desenvolvimento de Software

Como já foi dito, para que um desenvolvimento de software atinja a qualidade desejada é necessária a aplicação de um processo de desenvolvimento. Segundo a Wikipedia (2007):

> "Um processo de desenvolvimento de software é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de software. É estudado dentro da área de Engenharia de Software, sendo considerado um dos principais mecanismos para se obter software de qualidade e cumprir corretamente os contratos de desenvolvimento, sendo uma das respostas técnicas adequadas para resolver a Crise do software."

Segundo Pressman (2006), pode-se ter outra definição:

> "Os processos de software formam a base para o controle gerencial de projetos de software e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho são produzidos, marcos são estabelecidos, qualidade é assegurada e modificações são geridas."

A seção a seguir mostra, de forma simplificada, os modelos clássicos da Engenharia de Software, aceitos por Pressman (2006), Paula (2001), Larman (2002) e Sommerville (2003).

2.2.1 Modelo Clássico

O ciclo de vida clássico, conhecido como seqüencial linear ou em cascata, é o modelo mais antigo e mais amplamente usado na engenharia de software (Figura 2.5). Requer uma abordagem sistemática, seqüencial ao desenvolvimento de software (que se inicia no nível do sistema e avança ao longo da análise, projeto, codificação, testes e manutenção). O ciclo de vida clássico, porém, apresenta um grave problema. Existe a necessidade de se estabelecer todos os requisitos na fase de análise, fato este que em geral é difícil tanto para o cliente quanto para o desenvolvedor, já que os requisitos mudam constantemente. Outro problema é a demora para apresentação de uma versão executável do software.

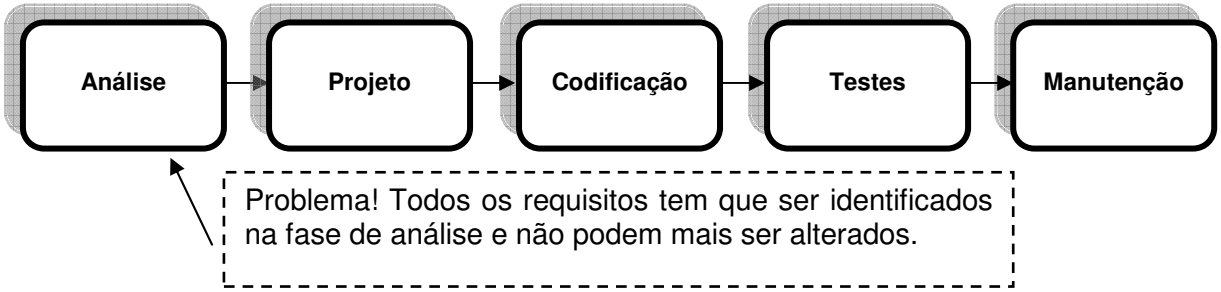

FIGURA 2.5 – Modelo Seqüencial Linear ou Cascata (Fonte: Adaptado de SOMMERVILLE, 2003).

2.2.2 Modelo de Prototipagem

O ciclo de vida de Prototipagem é um processo que possibilita que o desenvolvedor crie um modelo do software que deve ser construído para uma prévia avaliação tanto do cliente quanto do desenvolvedor. O modelo de prototipagem serve então como um importante mecanismo de identificação de requisitos. Este modelo passa pelos seguintes processos: obtenção dos requisitos (cliente e desenvolvedor definem os objetivos gerais do software); projeto rápido (abordagens de entrada e formatos de saída); construção do protótipo (implementação do projeto rápido), e; avaliação do protótipo (cliente e desenvolvedor avaliam o protótipo). Após esta avaliação, os requisitos são refinados, retornando ao processo 1 até que todos os requisitos sejam identificados. Após a identificação de todos requisitos, idealmente, o protótipo então é descartado, já que não foi construído observando-se nenhuma técnica de desenvolvimento para a garantia da qualidade, e o produto final é construído, agora sim com foco na qualidade.

A Figura 2.6 mostra claramente o ciclo de desenvolvimento do modelo de prototipagem, enfatizando a sua forma cíclica até a definição de todos os requisitos necessários para o desenvolvimento do software.

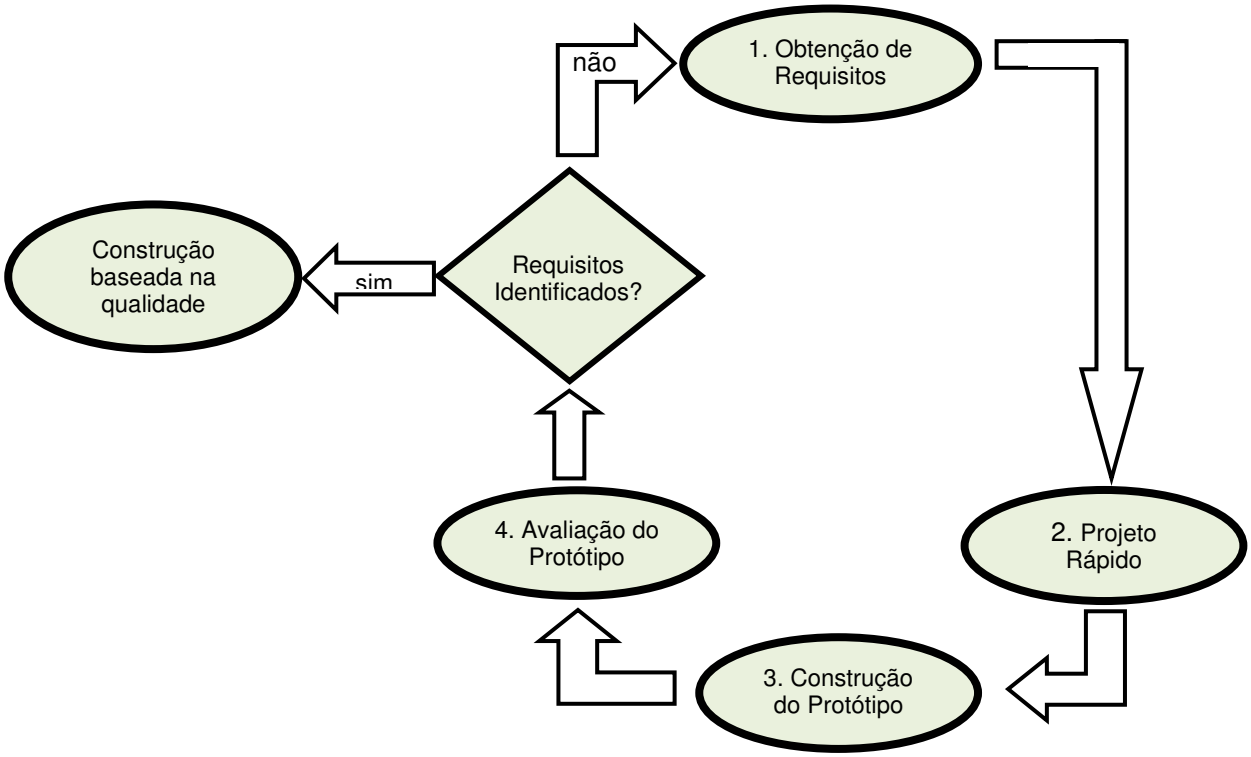

FIGURA 2.6 – Modelo de Prototipagem.

2.2.3 Modelo RAD

O modelo RAD (Rapid Application Development) caracteriza-se por ser um processo onde há o desenvolvimento rápido de uma aplicação (Figura 2.7). É um processo de desenvolvimento de software incremental que possui um ciclo de vida extremamente curto. Este modelo de desenvolvimento tem aplicação direta nos casos em que o sistema pode ser modularizado. Neste caso, cada uma das equipes de desenvolvimento fica responsável por um módulo, enquanto que outras equipes desenvolvem outros módulos, de forma concorrente. Ao final da construção de cada módulo há uma integração entre eles. Este ciclo se repete até que o software esteja completamente pronto. Este modelo possui cinco fases. A fase de modelagem do negócio visa a organização dos requisitos do sistema a ser construído. A fase de modelagem dos dados identifica as características e relações entre objetos de dados. Na fase de modelagem do processo, os objetos de dados definidos anteriormente são transformados para conseguir o fluxo de informação necessário para implementar uma função do negócio. Já na fase de geração da aplicação, utiliza-se ferramentas de quarta geração, que reutilizam componentes de programa já existentes ou criam novos componentes reutilizáveis. Por fim, na fase de teste e entrega, os componentes e interfaces são testados antes da entrega ao cliente.

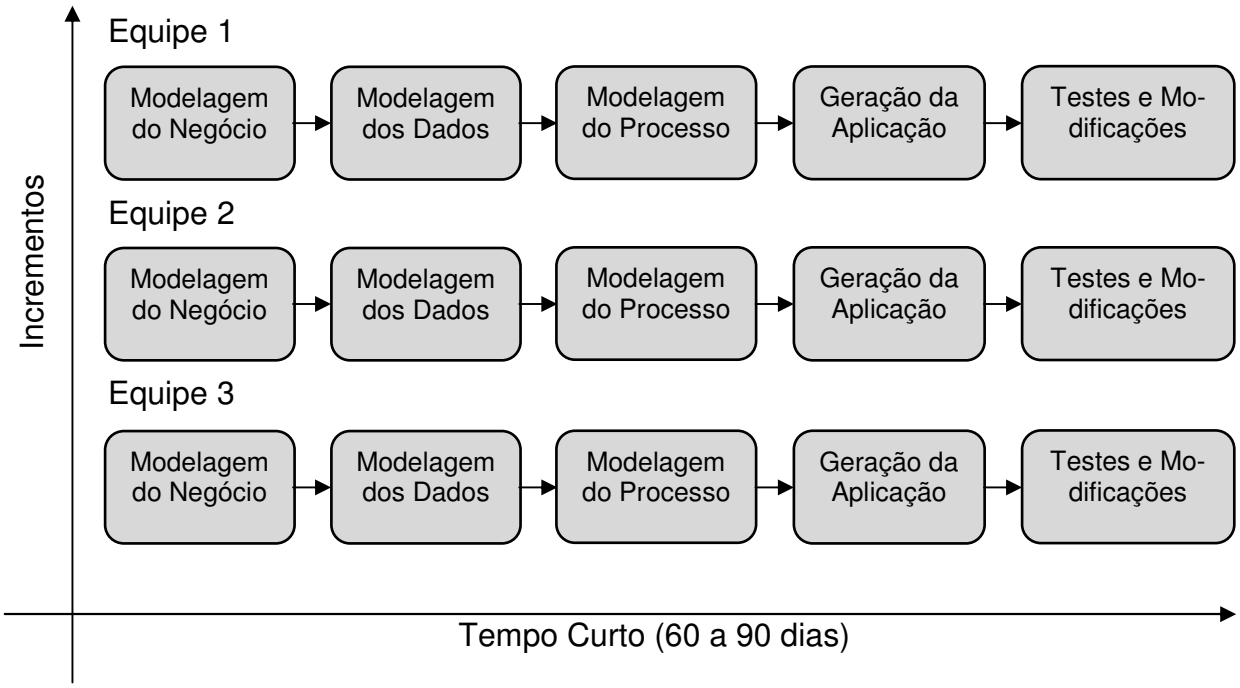

FIGURA 2.7 – Modelo RAD (Fonte: Adaptado de PRESSMAN, 2006).

2.2.4 Modelo Incremental

O modelo incremental segue a filosofia de refinamento e/ou incremento de funcionalidades do software, onde gera-se primeiramente um produto com requisitos básicos e a partir daí são gerados outros produtos cada vez mais detalhados até que se tenha o software completo. Este modelo é um melhoramento do modelo clássico, pois permite a alteração dos requisitos durante o desenvolvimento do software, que era um problema grave daquele modelo. A Figura 2.8 apresenta o modelo incremental.

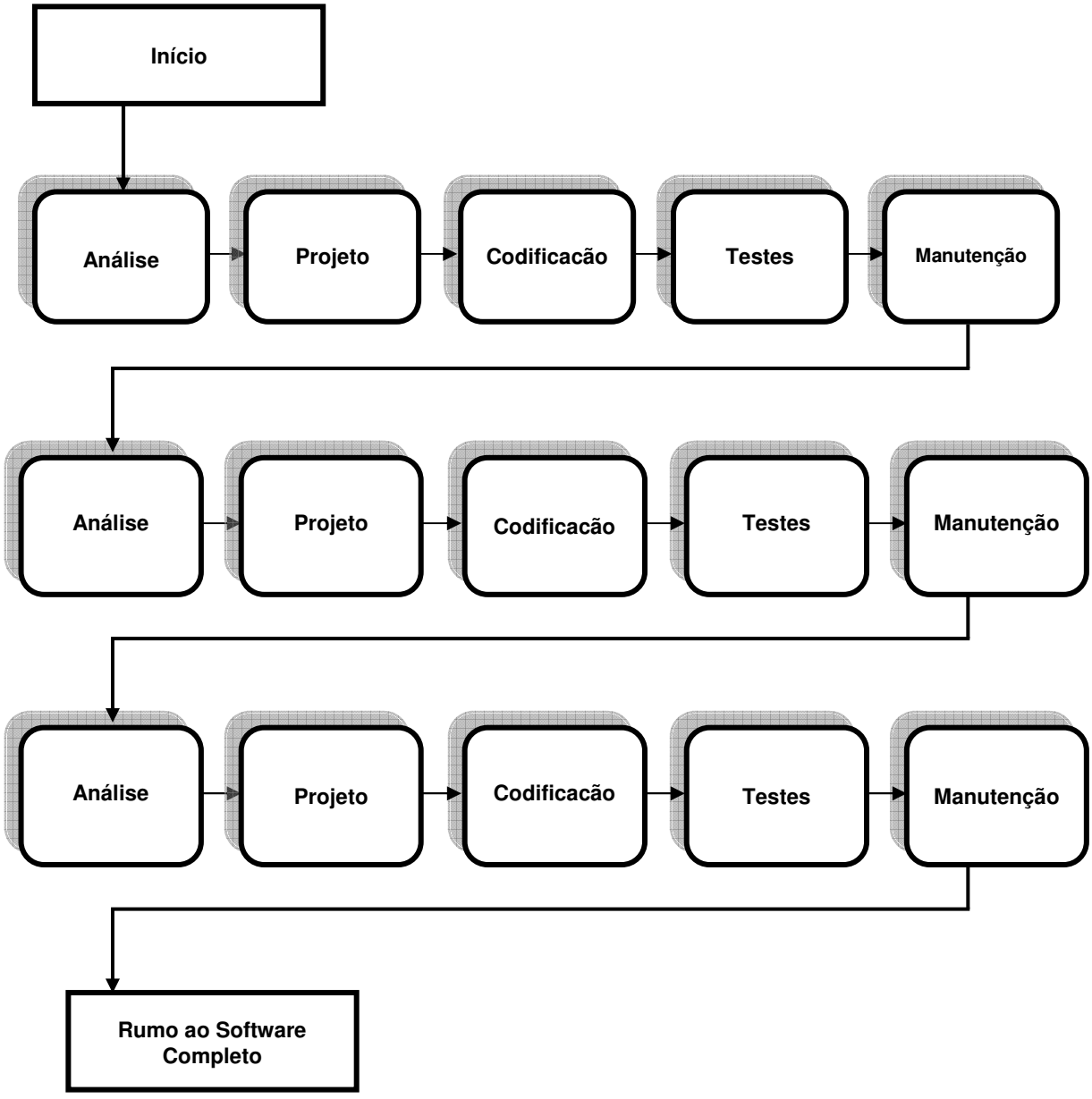

FIGURA 2.8 – Modelo Incremental (Fonte: Adaptado de SOMMERVILLE, 2003).

2.2.5 Modelo Espiral

O modelo espiral (Figura 2.9) divide o processo de construção de software em fases que são: comunicação com o cliente, planejamento, análise de riscos, engenharia, construção e liberação e avaliação pelo cliente. Iniciado o processo, ele passa por todas as fases e depois volta à primeira novamente, executando novamente todas as fases como se fosse uma espiral até que se tenha o produto desejado. Atualmente é a abordagem mais realística para o desenvolvimento de software em grande escala, exatamente pelo feedback que é conseguido na fase de avaliação do cliente. Principalmente nesta fase, as falhas do software são identificadas e corrigidas, impedindo assim que estas falhas se propaguem para as próximas iterações do ciclo de desenvolvimento do software. Além disso, existe uma fase que é a análise de riscos. Nesta fase, que é posterior à fase de planejamento, os riscos da nova implementação são avalidados, para que não se tenha um desenvolvimento inviável na próxima etapa do modelo.

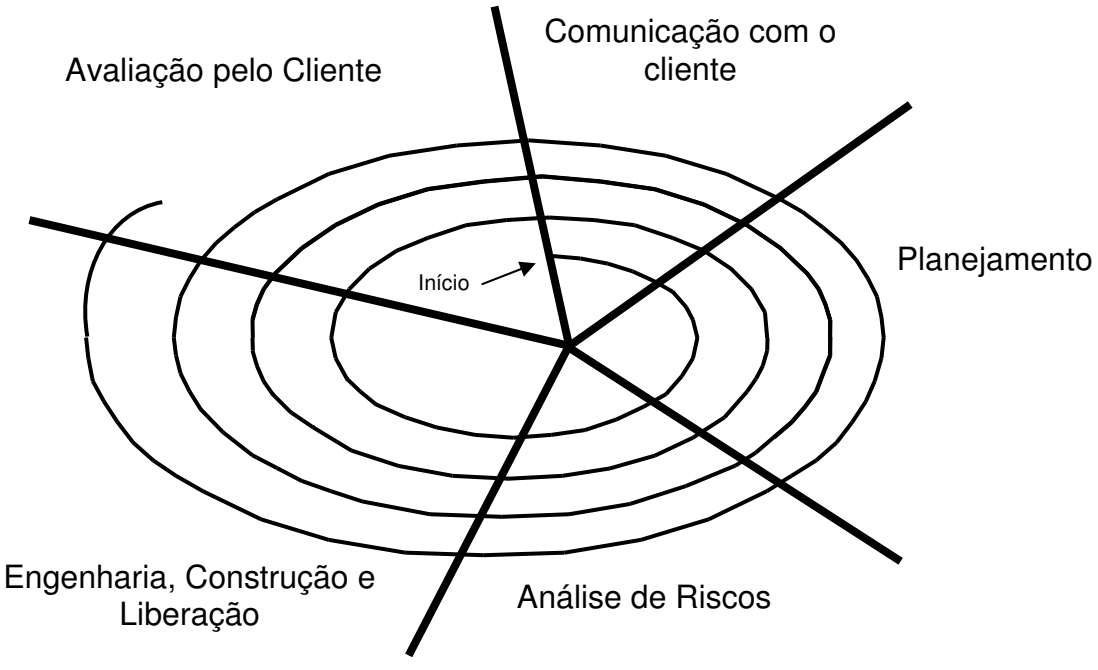

FIGURA 2.9 – Modelo Espiral (Fonte: Adaptado de SOMMERVILLE, 2003).

2.3 UML – Unified Modeling Language

A UML foi criada em meados da década de 1990, por Grady Booch, Ivar Jacobson e James Rambaugh. Cada um destes autores possuíam métodos particulares (Booch, OOSE e OMT, respectivamente). Quando perceberam que seus métodos estavam convergindo um em direção ao outro, de maneira independente, resolveram criar uma linguagem única de modelagem, que pudesse abranger todas as características principais dos métodos particulares de cada um (Booch *et. al.*, 2000; Guedes, 2006).

Ainda segundo Booch et al.(2000), a UML pode ser definida como:

> "uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de software. A UML proporciona uma forma-padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, esquema de banco de dados e componentes de software reutilizáveis."

Como a UML é voltada para o desenvolvimento de sistemas orientados a objetos, alguns conceitos necessários sobre este tópico são apresentados na sub-seção seguinte:

2.3.1 Conceitos de Orientação a Objetos

A orientação a objetos, também conhecida como Programação Orientada a Objetos (POO) ou ainda em inglês Object-Oriented Programming (OOP) é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos (Wikipedia, 2007).

Booch *et. al.* (2000), Pressman (2006), Paula (2001) e Wikipedia (2007) apresentam os conceitos fundamentais da orientação a objetos. Estes conceitos são mostrados na Tabela 2.1, de forma resumida:

TABELA 2.1 – Conceitos de Orientação a Objetos

Conceito	Breve Descrição
Classe	definição abstrata que representa um conjunto de objetos com características comuns.
Objeto	é a materialização de um conceito, mais especificamente de uma classe.
Atributos	são as características peculiares de um objeto.Os atributos armazenam valores determinados.
Métodos	definem as responsabilidades dos objetos.
Mensagem	é uma requisição enviada à um objeto, com o objetivo de invocar um de seus métodos, ativando assim um comportamento descrito por sua classe.
Sobrecarga	é a utilização do mesmo nome para símbolos ou métodos com operações ou funcionalidades distintas. Geralmente diferencia-se os métodos pela sua assinatura.
Herança	é o mecanismo pelo qual uma classe (sub-classe) pode estender outra classe (super-classe), aproveitando seus comportamentos (métodos) e estados possíveis (atributos).
Associação	é o mecanismo pelo qual um objeto utiliza os recursos de outro ou possui uma ligação direta com este.
Agregação	A agregação é um caso particular da associação. A agregação indica que uma das classes do relacionamento é uma parte, ou está contida em outra classe.
Generalização	A generalização é um relacionamento entre um elemento geral e um outro mais específico. O elemento mais específico possui todas as características do elemento geral e contém ainda mais particularidades. Um objeto mais específico pode ser usado como uma instância do elemento mais geral.
Dependência	O relacionamento de dependência é uma conexão semântica entre dois modelos de elementos, um independente e outro dependente. Uma mudança no elemento independente irá afetar o modelo dependente.
Encapsulamento	este mecanismo é utilizado amplamente para impedir o acesso direto aos atributos de um objeto, disponibilizando externamente apenas os métodos que alteram estes estados.
Polimorfismo	é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura (lista de parâmetros e retorno) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.
Interface	é o meio de comunicação entre a classe e o meio externo a ela. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.

2.3.2 Diagramas da UML 2.0

A UML é composta por vários diagramas, cada um com aplicação específica em uma determinada fase do desenvolvimento de um software. Os diversos diagramas existentes na UML podem ser vistos, agrupados em diagramas estruturais e diagramas comportamentais, na Figura 2.10.

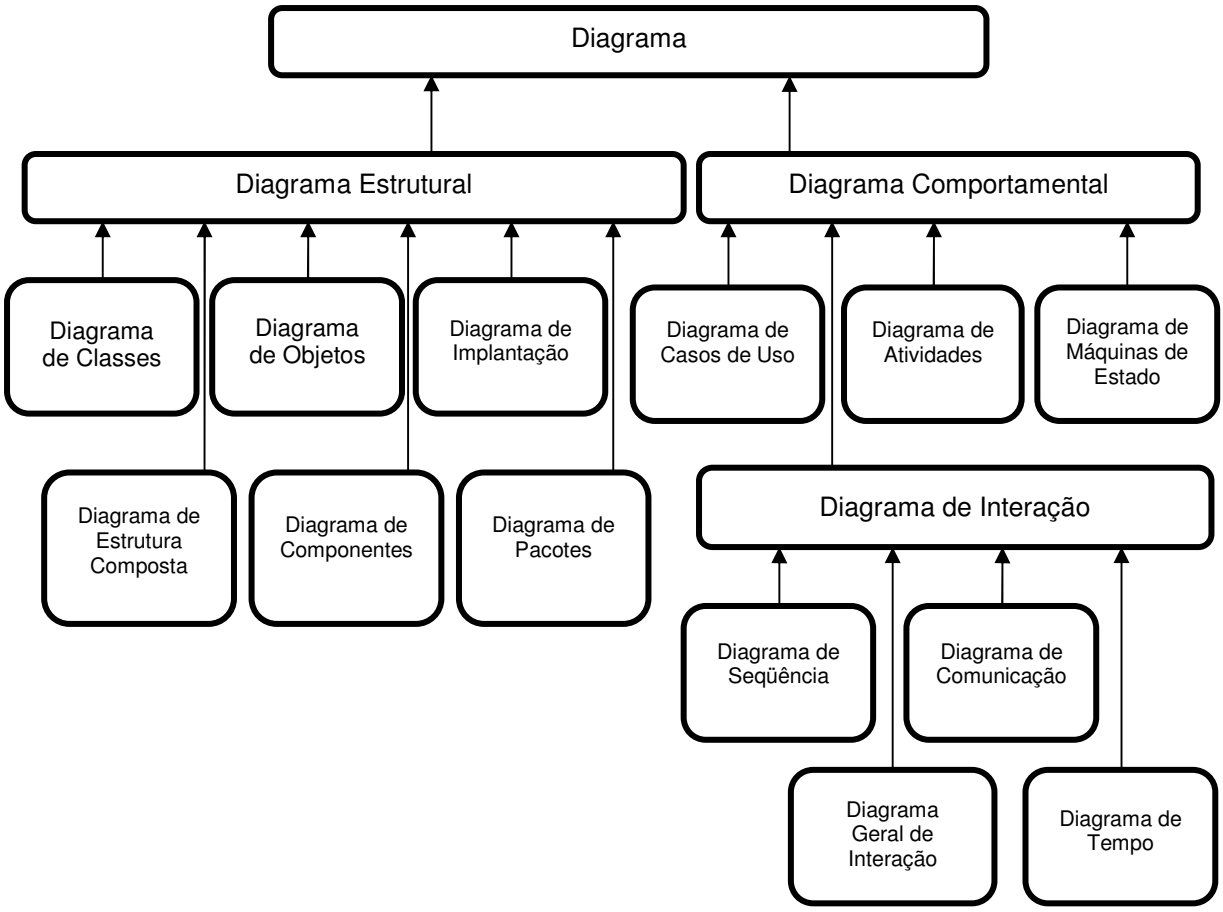

FIGURA 2.10 – Diagramas da UML 2.0 (Fonte: Adaptado de GUEDES, 2006)

As seções seguintes apresentam uma breve descrição dos diagramas da UML 2.0, com base em Booch *et. al*. (2000), Guedes (2006) e Shalloway e Trott (2004).

Cada um dos diagramas apresenta um exemplo gráfico, com explicação textual em seguida. Todos os diagramas se referem à modelagem de um software científico que trata do Bin Packing Problem.

2.3.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso é utilizado na fase de levantamento e análise de requisitos. Este diagrama apresenta uma visão geral do sistema que será desenvolvido, demostrando quais as funções que cada elemento (usuário, outros sistemas e até mesmo, um hardware), chamado de ator, desempenha dentro do

sistema. A Figura 2.11 retrata um diagrama de casos de uso de um software científico.

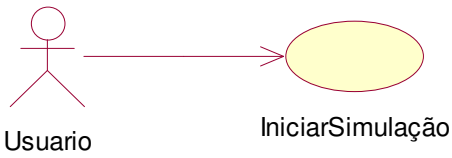

FIGURA 2.11 – Diagrama de Casos de Uso

Neste exemplo, existe apenas uma interface com o software, chamada de Usuário, e este, por sua vez, tem apenas a opção de iniciar a simulação.

2.3.2.2 Diagrama de Classes

É o diagrama mais amplamente utilizado dentro da UML. Como o próprio nome define, este diagrama mostra a estrutura das classes dentro de um sistema orientado a objetos. Este diagrama conta com uma grande quantidade de elementos gráficos que podem ser utilizados para demonstrar o relacionamento entre as classes. A Figura 2.12 mostra um diagrama de classes e seus relacionamentos. Neste exemplo são mostradas três classes que se relacionam através de associações simples.

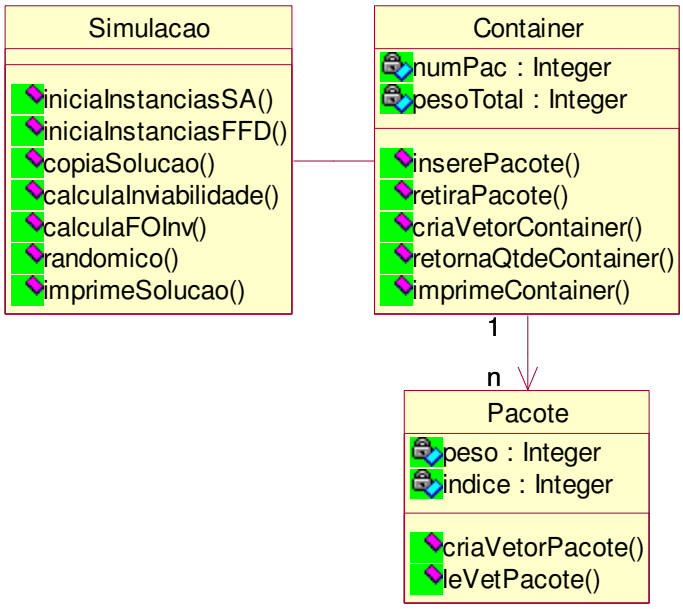

FIGURA 2.12 – Diagrama de Classes

2.3.2.3 Diagrama de Objetos

O diagrama de objetos é praticamente idêntico ao diagrama de classes, com a diferença de retratar objetos e não classes. Em outras palavras, o diagrama de objetos fornece uma visão dos valores armazenados pelos objetos, derivados de um diagrama de classes, em um determinado momento da execução de um processo executável. Um exemplo de diagrama de objetos pode ser visto na Figura 2.13. Neste exemplo dois objetos se relacionam.

FIGURA 2.13 – Diagrama de Objetos

2.3.2.4 Diagrama de Estrutura Composta

Segundo Guedes (2006), este é um dos três novos diagramas propostos pela UML 2.0. Este diagrama descreve a estrutura interna de um classificador, como uma classe ou componente, detalhando as partes internas que o compõem, como estas se comunicam e colaboram entre si. Também é utilizado para descrever uma colaboração onde um conjunto de instâncias cooperam entre si para realizar uma tarefa. A Figura 2.14 mostra um exemplo deste diagrama. Este exemplo mostra a cooperação entre partes do sistema.

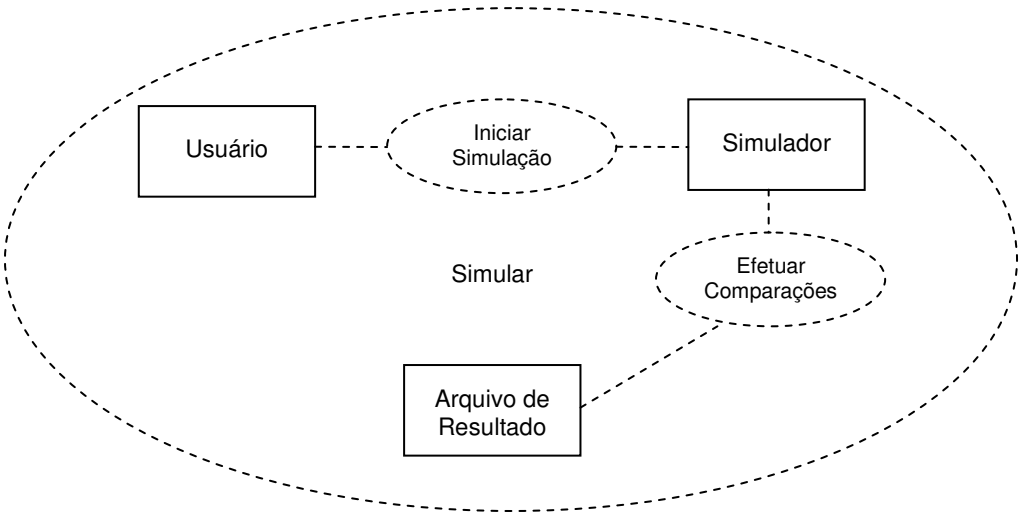

FIGURA 2.14 – Diagrama de Estrutura Composta

2.3.2.5 Diagrama de Seqüência

Este é um dos diagramas que retratam a interação e o comportamento do sistema. Este diagrama preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Em geral este diagrama apresenta os atores identificados no diagrama de casos de uso, além das classes identificadas no diagrama de classes, mostrando interação e a troca de mensagens entre eles. Um exemplo deste diagrama pode ser visto na Figura 2.15. A seqüência de ações e mensagens é claramente notada neste diagrama.

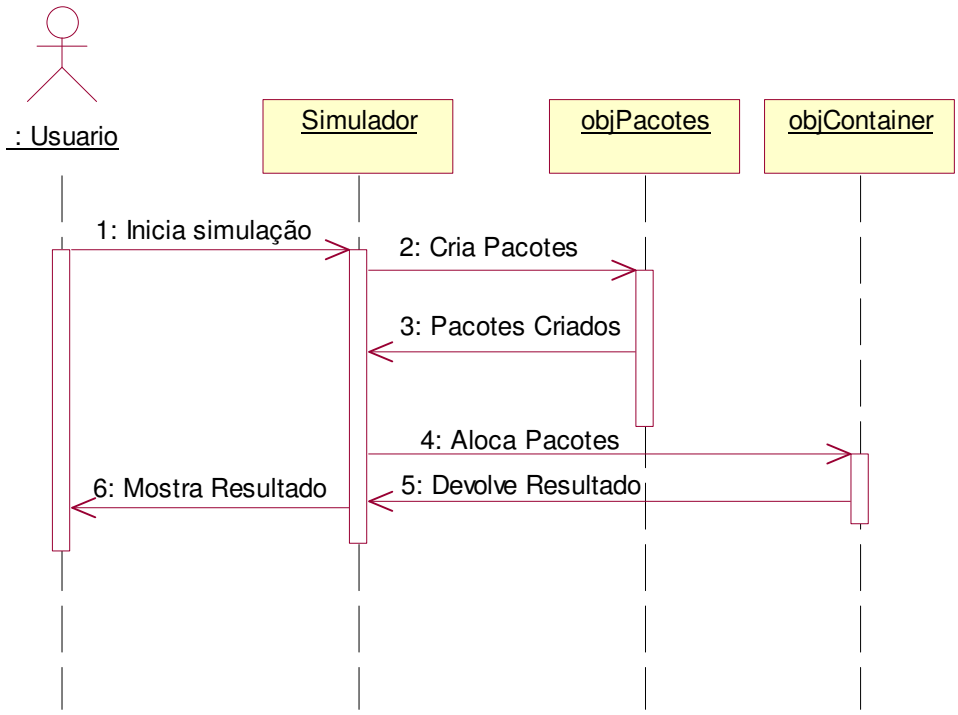

FIGURA 2.15 – Diagrama de Seqüência

2.3.2.6 Diagrama de Comunicação

Segundo Guedes (2006), o diagrama conhecido como diagrama de colaboração, até a versão 1.5 da UML, sofreu uma modificação no seu nome e passou a se chamar diagrama de comunicação. Este diagrama está amplamente associado ao diagrama de seqüência. No entanto, este diagrama não mostra o tempo em que acontece a interação entre os objetos, mas se concentra em como os objetos estão vinculados e quais mensagens eles trocam entre si durante o processo. A Figura 2.16 retrata um diagrama de comunicação, com as mensagens trocadas entre os objetos.

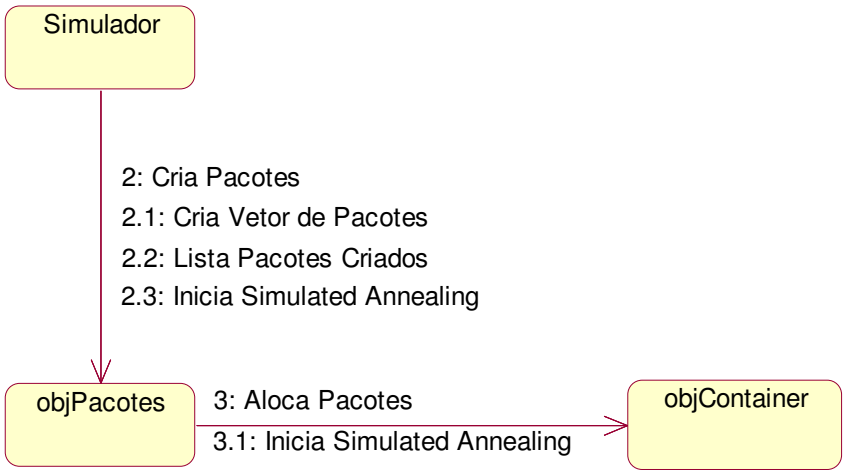

FIGURA 2.16 – Diagrama de Comunicação

2.3.2.7 Diagrama de Máquinas de Estado

Ainda segundo Guedes (2006), o diagrama conhecido como diagrama de estados, até a versão 1.5 da UML, sofreu uma modificação no seu nome e passou a se chamar diagrama de máquinas de estado. Este diagrama é utilizado, principalmente, para demonstrar os estados por que passa uma determinado processamento dentro do sistema. Uma característica que geralmente é encontrada neste diagrama é o uso de gerúndio para identificar o estado, para enfatizar as transições entre eles. Um diagrama de máquinas de estados pode ser observado na Figura 2.17. Nesta figura são retratados alguns estados assumidos pelo sistema ao longo do seu processamento.

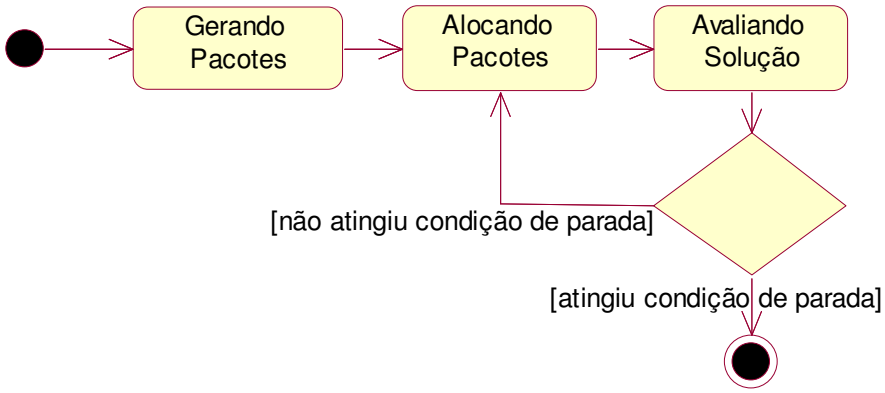

FIGURA 2.17 – Diagrama de Máquinas de Estados

2.3.2.8 Diagrama de Atividades

Esse diagrama assemelha-se com um fluxograma, onde pode-se notar o fluxo seguido pela aplicação, desde seu início até o seu término, indicados, respectivamente, por uma bolha totalmente preenchida e por outra bolha preenchida circundada. Este diagrama inclui estruturas condicionais de desvio de fluxo de processamento. Um exemplo de diagrama de atividades é mostrado na Figura 2.18. Neste diagrama são mostradas as atividades que são executadas ao longo da execução do software.

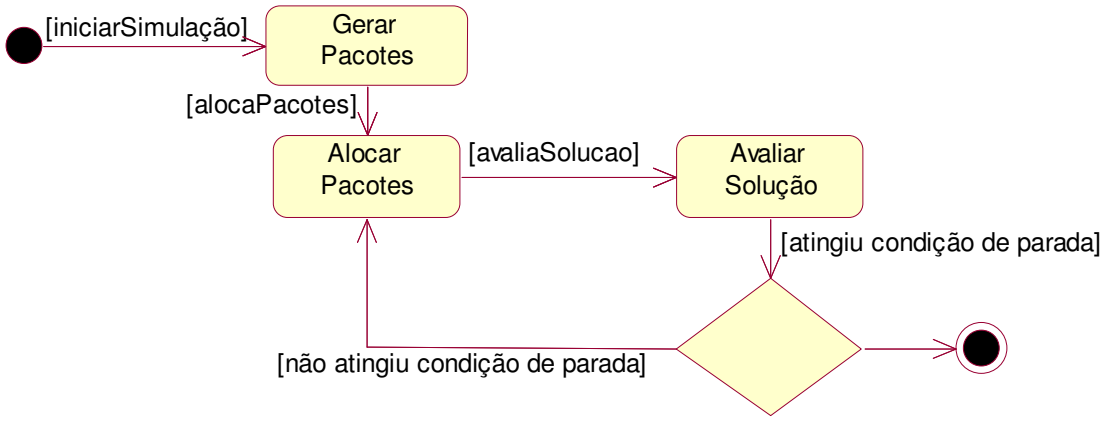

FIGURA 2.18 – Diagrama de Atividades

2.3.2.9 Diagrama de Componentes

Este diagrama está intimamente ligado à linguagem de programação utilizada na concepção do sistema. Ele retrata a interação ente os módulos de código-fonte, bibliotecas e demais arquivos em geral que interagirão para permitir que o sistema funcione. A Figura 2.19 retrata um diagrama de componentes. Os componentes que compõem o simulador estão mostrados neste diagrama.

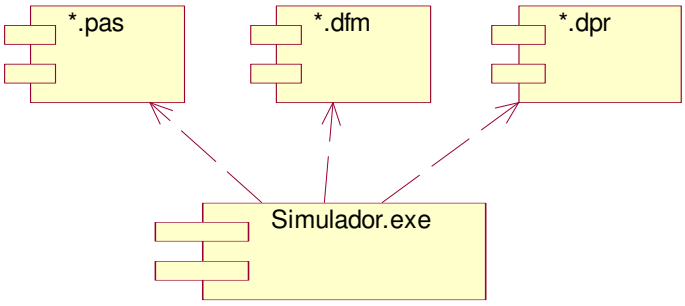

FIGURA 2.19 – Diagrama de Componentes

2.3.2.10 Diagrama de Implantação

Este diagrama retrata a interação entre servidores, estações, topologias, protocolos de comunicação e demais características físicas necessárias para o perfeito funcionamento da aplicação. Um diagrama de implantação pode ser visto na Figura 2.20. O exemplo mostra dois computadores executando o simulador através de uma rede.

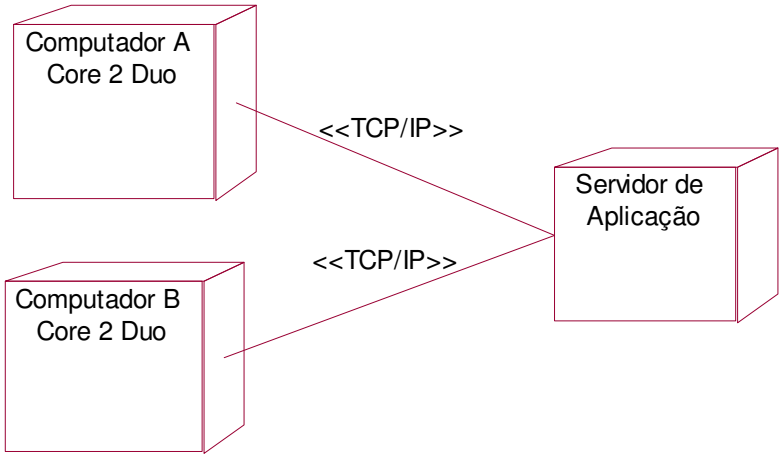

FIGURA 2.20 – Diagrama de Implantação

2.3.2.11 Diagrama de Pacotes

Com o uso deste diagrama, o desenvolvedor tem a possibilidade de encapsular os detalhes de um sub-sistema, mostrando apenas a sua interação com outros sub-sistemas, de forma a representar um entendimento melhor da aplicação, com um nível menor de detalhes. A Figura 2.21 mostra um exemplo de diagrama de pacotes.

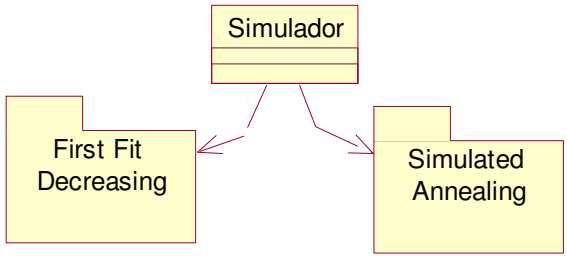

FIGURA 2.21 – Diagrama de Pacotes

2.3.2.12 Diagrama Geral de Interação

Segundo Guedes (2006), este diagrama passou a existir na proposta da UML 2.0. Este diagrama é uma variação do diagrama de atividades e fornece uma visão global dentro de um sistema. Um exemplo deste diagrama pode ser visto na Figura 2.22.

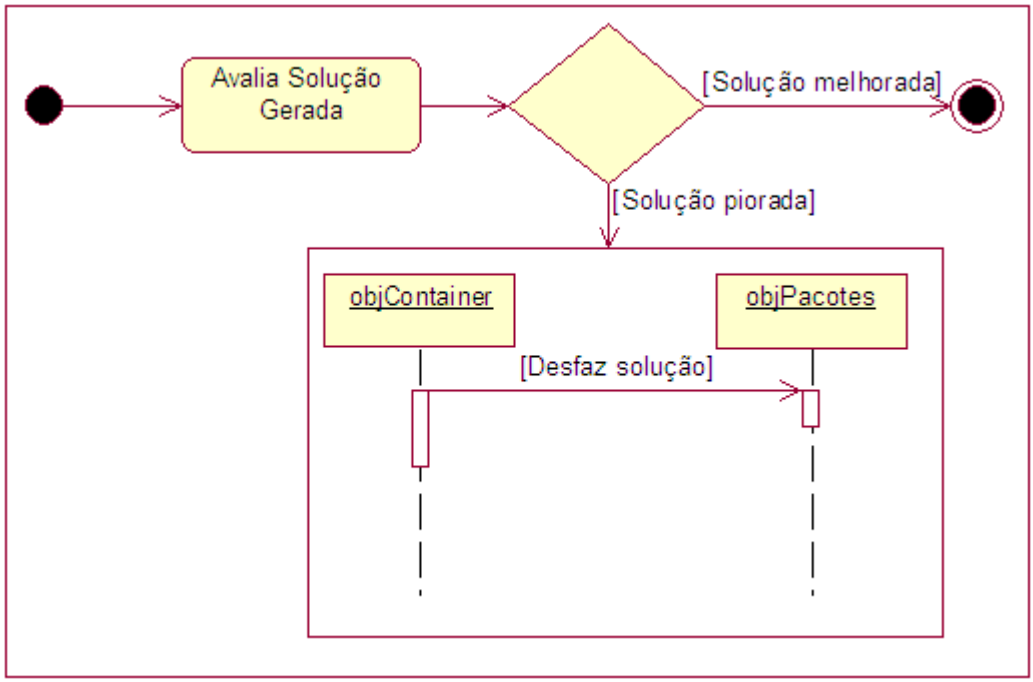

FIGURA 2.22 – Diagrama Geral de Interação (Fonte: GUEDES, 2006)

2.3.2.13 Diagrama de Tempo

Ainda segundo Guedes (2006), este é o terceiro diagrama que passou a existir na proposta da UML 2.0. Este diagrama descreve a mudança no estado ou condição de uma instância de uma classe ou seu papel durante um tempo. É tipicamente utilizado para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos. A Figura 2.23 retrata o novo diagrama que passa a fazer parte da UML 2.0.

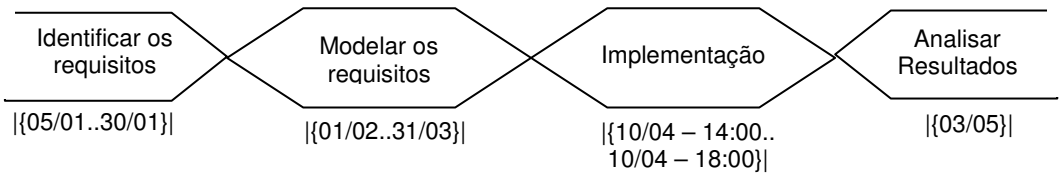

FIGURA 2.23 – Diagrama de Tempo

2.4 Processos de Desenvolvimento de Software

2.4.1 Processo Unificado

A seguir é feita uma explanação sobre o Processo Unificado (PU), baseada em Sommerville (2003), Pressman (2006), Jacobson (1999), Larman (2004) e Wazlawick (2004).

O Processo Unificado (PU) surgiu como um processo popular para o desenvolvimento de software visando à construção de sistemas orientados a objetos. Este processo utiliza a UML como notação gráfica básica. É um processo iterativo e incremental, dirigido por casos de uso e centrado na arquitetura.

O Processo Unificado propõe um processo ágil, com poucos artefatos e pouca burocracia, o qual permite o desenvolvimento de software rapidamente.

Como já foi dito, o PU utiliza o modelo evolutivo (Figura 2.24). Em cada iteração do seu ciclo de desenvolvimento, são gerados incrementos sucessivos, convergindo para a versão completa do sistema. Cada iteração possui suas próprias atividades de análise de requisitos, projeto, implementação e testes. Um ponto determinante no Processo Unificado é que, a cada iteração, é gerado um produto final de qualidade, e não um protótipo.

2.4.1.1 Fases do Processo Unificado.

O processo unificado comporta, em suas recomendações, as antigas fases de estudo de viabilidade, análise de requisitos, análise de domínio e o projeto em múltiplas camadas. Contudo, estas fases aparecem no Processo Unificado de forma diferente. O Processo Unificado organiza suas iterações em quatro fases principais, resumidas na Tabela 2.1 e também retratadas na Figura 2.24.

TABELA 2.1 – Fases do Processo Unificado. (Fonte: PAULA, 2001).

Fase	Descrição
Concepção	Fase na qual se justifica a execução de um projeto de desenvolvimento de software, do ponto de vista do negócio do cliente.
Elaboração	Fase na qual o produto é detalhado o suficiente para permitir um planejamento acurado da fase de construção.
Construção	Fase na qual é produzida uma versão completamente operacional do produto.
Transição	Fase na qual o produto é colocado à disposição de uma comunidade de usuários.

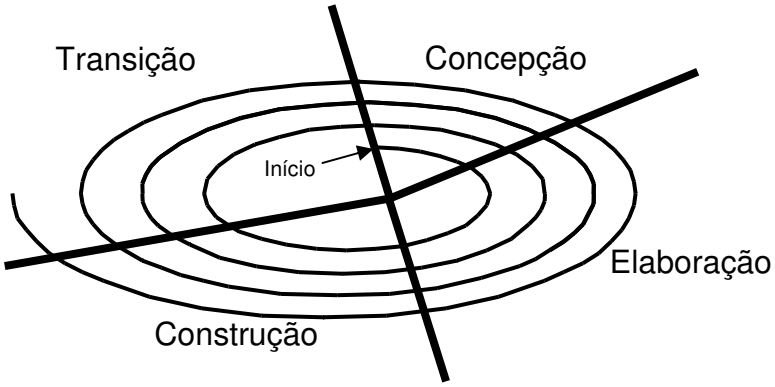

FIGURA 2.24 – Fases do Processo Unificado.

O ciclo evolutivo adotado pelo Processo Unificado permite um controle real sobre a natureza mutável dos requisitos, uma vez que são feitos planejamentos de pequenas partes do sistema para cada iteração. Como se tem uma parte reduzida do sistema em determinada iteração, os requisitos também reduzem consideravelmente, o que os torna mais fácil de administrar e tratar. Outra vantagem do ciclo evolutivo é o contato rápido do cliente com uma versão executável do sistema, o que permite o seu feedback, que é conseguido ao final de cada iteração. Esta identificação precoce das opiniões do usuário permite a correção do curso do desenvolvimento, de forma que ao seu final não existam falhas. A aplicação ideal do PU prega que as iterações devem apresentar um curto período de duração, pois são mais gerenciáveis e permitem rápida realimentação. O Processo Unificado possui ainda alguns fluxos que são explicados na Tabela 2.2.

TABELA 2.2 – Fluxos do Processo Unificado. (Fonte: PAULA, 2001).

Fase	Descrição
Requisitos	Fluxo que visa obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Análise	Fluxo cujo objetivo é detalhar, estruturar e validar os requisitos, de forma que esses possam ser usados como base para o planejamento detalhado.
Desenho	Fluxo cujo objetivo é formular um modelo estrutural do produto que sirva de base para a implementação.
Implementação	Fluxo cujo objetivo é realizar o desenho em termos de componentes de código
Testes	Fluxo cujo objetivo é verificar os resultados da implementação.

2.4.2 RUP (Rational Unified Process)

O RUP (Rational Unified Process) é um produto desenvolvido e comercializado pela empresa Rational que oferece os detalhes necessários para executar projetos de software a partir do processo unificado, incluindo orientações, modelos e ferramental. Essencialmente, o RUP é um produto comercial baseado no Processo Unificado.

Segundo Krutchen (2003) e Rezende (2005), o RUP pode ser considerado como:

- uma abordagem de desenvolvimento de software dirigida por casos de uso, iterativa e centrada na arquitetura.
- um processo de engenharia de software bem definido e bem estruturado. Ele claramente define quem é o responsável pelo que, como as coisas são feitas e quando fazê-las.
- um processo que fornece um framework de processo customizável para a engenharia de software. Essas customizações podem ser feitas para suportar pequenas equipes e abordagens disciplinadas ou menos formal para o desenvolvimento.

Ainda segundo Rezende (2005), o RUP possui alguns elementos básicos, que são apresentados na Tabela 2.3.

TABELA 2.3 – Elementos Básicos do RUP.

Elemento	Descrição
Workers	Definem o comportamento e as responsabilidades dos indivíduos dentro da equipe multidisciplinar do projeto. Atuam como proprietários de um conjunto de artefatos e desenvolvem um conjunto de atividades
Atividades	Correspondem à criação ou utilização de artefatos onde cada atividade é atribuída a um worker específico.
Artefatos	São processos, atividades ou informações produzidas, modificadas ou utilizadas pelo processo como produtos tangíveis do projeto.
Workflows	São seqüências de atividades que produzem resultados com valor observável onde mostram as interações entre os workers, tais como diagramas ou descrições de requisitos funcionais.

2.4.2.1 Fases do RUP

O RUP possui as seguintes fases, segundo Wazlawick (2004) e RUP(2007):

- **Concepção (Inception)**: foca no estabelecimento do escopo e visão do projeto, ou seja, visa a compreensão do problema e da tecnologia por meio da definição dos casos de uso mais críticos.
- **Elaboração (Elaboration)**: foca no estabelecimento dos requisitos do sistema e na sua arquitetura, isto é, estabelece um plano de projeto e uma arquitetura sólida.
- **Construção (Construction)**: foca na completa construção do sistema, incluindo ainda a preparação dos clientes. Ainda nesta fase são feitos os testes necessários no sistema.
- **Transição (Transition)**: foca na completa transição ou entrega do sistema para os usuários. Visa também fornecer treinamento aos usuários finais.

2.4.2.2 Ciclo de Desenvolvimento

O ciclo de desenvolvimento do RUP é o ciclo conhecido como espiral, que permite a abordagem incremental. Tal abordagem define uma estratégia de desenvolvimento interativo, na qual o sistema é construído em partes pequenas, uma após a outra, até o seu término.

2.4.2.3 As melhores práticas

Krutchen (2001) define algumas práticas que devem ser adotadas, visando a melhoria da qualidade dos produtos gerados, bem como um maior controle sobre o seu desenvolvimento. São elas:

Desenvolvimento iterativo: permite um desenvolvimento mais realista, pois divide o projeto em ciclos menores, conhecidos como iterações, sendo que em cada uma delas são executadas todas as fases do desenvolvimento, gerando assim uma versão executável.

Gerência dos requisitos: a característica mutável dos requisitos é certa. Esta prática busca gerenciar estas mudanças, para que a qualidade do produto não seja alterada por este motivo.

Uso de arquiteturas baseadas em componente: um sistema deve ser desenvolvido utilizando componentes já prontos, certificados e testados, o que caracteriza reuso de código. Esta prática oferece grandes vantagens como a manutenção localizada.

Modelo visual do software: a adoção de modelos gráficos como uma notação padronizada permite um entendimento geral da equipe, o que permite uma maior clareza na construção do sistema.

Verificação constante da qualidade do software: com o controle constante de qualidade, a detectação precoce de uma falha acontece de forma natural, o que impede a sua propagação para as próximas etapas do desenvolvimento.

Controle de Mudanças do Software: visa controlar e padronizar o desenvolvimento, para garantir a integridade na atualização dos produtos de trabalho.

2.4.3 XP (Extreme Programming)

Segundo Beck (2000), os processos ágeis, como o Extreme Programming, aplicam-se com especial relevância em pequenos projetos ou projetos com equipes de trabalho co-localizadas. Apresentam uma visão semelhante sobre as boas práticas necessárias ao desenvolvimento e software de qualidade, como, por exemplo, o desenvolvimento iterativo e a preocupação nos requisitos e envolvimento dos utilizadores finais.

Os conceitos sobre a Extreme Programming mostrados abaixo são baseados em Astels et.al (2002), Jeffries (2007) e Beck (2000).

Extreme Programming (XP) é um processo de desenvolvimento que possibilita a criação de software de alta qualidade, de maneira ágil, econômica e flexível. Vem sendo adotado com enorme sucesso na Europa, nos Estados Unidos e, mais recentemente, no Brasil.

Cada vez mais as empresas convivem com ambientes de negócios que requerem mudanças freqüentes em seus processos, as quais afetam os projetos de software. Os processos de desenvolvimento tradicionais são caracterizados por uma grande quantidade de atividades e artefatos que buscam proteger o software contra mudanças, o que faz pouco ou nenhum sentido, visto que os projetos devem se adaptar a tais mudanças ao invés de evitá-las. A XP concentra os esforços da equipe de desenvolvimento em atividades que geram resultados rapidamente na forma de software intensamente testado e alinhado às necessidades de seus usuários. Além disso, simplifica e organiza o trabalho combinando técnicas comprovadamente eficazes e eliminando atividades redundantes. Por fim, reduz o risco dos projetos desenvolvendo software de forma iterativa e reavaliando permanentemente as prioridades dos usuários.

Existem algumas práticas que são adotadas neste processo de desenvolvimento de software voltado para o desenvolvimento de software comerciais. Segundo Astels *et. al* (2002), as práticas da Extreme Programming são criadas para funcionar juntas e fornecer mais valor do que cada uma poderia fornecer individualmente.
A XP apresenta uma grande preocupação com relação a alteração constante dos requisitos do sistema. Sobre esta questão, a XP é enfática; o desenvolvedor deve permitir que o projeto seja flexível. Em outras palavras, o desenvolvedor deve aceitar as alterações e não lutar contra elas.

A seção seguinte detalha os princípios definidos pela Extreme Programming, segundo definições de Astels *et. al* (2002).

2.4.3.1 Princípios da XP

A Extreme Programming (XP) define 13 princípios básicos, que devem ser adotados por qualquer projeto que se baseie nesta técnica. Estes 13 princípios são mostrados

na Tabela 2.4, juntamente com uma explicação resumida. Uma descrição mais detalhada de cada princípio é mostrada logo em seguida, nas seções seguintes deste trabalho.

TABELA 2.4 – Princípios da XP.

Princípio	Descrição resumida
1. Cliente faz parte da equipe de desenvolvimento	O cliente deve participar do desenvolvimento, fornecendo o feedback necessário para a correção precoce de falhas
2. Uso de metáforas	Deve-se utilizar metáforas para a definição de termos complexos, para permitir um conhecimento uniforme de toda a equipe.
3. Planejamento	O projeto deve ser constantemente planejado para permitir uma real identificação do seu progresso.
4. Reuniões curtas	As reuniões devem ser curtas, de preferência sem o uso de cadeiras, para que se discuta os termos estritamente relevantes.
5. Teste contínuo	Deve-se efetuar testes antes mesmo da implementação de um código. Este teste precoce permite um controle otimizado sobre as falhas de um projeto.
6. Simplicidade	O projeto deve ser o mais simples possível.
7. Programação em Pares	A programação deve ser feita em pares. Enquanto um programador tem um foco no código que está desenvolvendo, o observador tem uma visão macro do processo, permitindo a identificação facilitada de falhas.
8. Padrão de Codificação	Todos as pessoas envolvidas no processo devem definir e seguir um mesmo padrão de codificação.
9. Propriedade coletiva sobre o código-fonte	Todas as pessoas devem ter acesso livre ao código-fonte, de modo que possa acrescentar melhorias e corrigir falhas identificadas.
10. Integração contínua	Deve-se efetuar uma contínua integração do novo código gerado, para que seja possível identificar precocemente falhas na integração deste novo código.
11. Refatoração contínua	O desenvolvedor deve alterar a estrutura interna sempre, de forma a melhorar a sua performance, sem alterar o funcionamento externo do código.
12. Concepção de pequenas versões	Devem ser concebidas pequenas versões de código de cada vez, para que os usuários tenham rapidamente um contato com uma versão executável do software e conseqüentemente consigam gerar o feedback necessário.
13. Jornada de Trabalho	A equipe de desenvolvimento deve ter uma jornada de trabalho flexível e leve, para que não exista fadiga e estresse dos seus membros.

2.4.3.1.1 Cliente faz parte da equipe de desenvolvimento.

Segundo este critério, o cliente deve participar ativamente do processo de desenvolvimento do software em questão. Ele deve ser um dos eventuais usuários do sistema. Além disso, o cliente deve representar as necessidades de seus colegas que também serão usuários do sistema. Ele deve ser capaz de responder qualquer questionamento da equipe de desenvolvimento. Deve também tomar decisões de relativas à prioridade de recursos, riscos e qualquer outra questão relacionada à concepção do software. A grande vantagem de se ter o cliente trabalhando junto

com a equipe técnica de desenvolvimento é o feedback constante proporcionado por ele. O cliente também auxilia na fase de testes do software, verificando se as funcionalidades implementadas correspondem aos reqüisitos identificados no início do desenvolvimento.

2.4.3.1.2 Uso de Metáforas

As metáforas devem ser usadas para descrever os conceitos difíceis. É desejável a utilização de metáforas em casos de grande carga de abstração, pois esta possibilita uma compreensão facilitada do ambiente, fazendo com que todas as pessoas envolvidas compartilhem de uma compreensão da visão global do sistema e do problema que deve ser solucionado.

2.4.3.1.3 Planejamento

O planejamento é outro princípio da XP. Todos os projetos deve ser planejados para que se possa fornecer uma compreensão mútua para todas as partes sobre quanto tempo, aproximadamente, levará o projeto. Também serve para indicar o porte do projeto. Este planejamento deve ser simples e rápido, porém deve apresentar todas as análises necessárias do projeto. Este planejamento deve indicar o que e como será feito, bem como os riscos associados ao projeto. O planejamento não deve ser feito para um período muito longo, pois quanto mais longo for, maiores são as chances de que ele seja impreciso. O cliente auxilia nesta fase. Ele define o escopo do sistema, suas prioridades, o conteúdo e as datas de lançamento de cada versão. Já a equipe técnica define as estimativas de custo e prazo e o processo que será utilizado.

2.4.3.1.4 Reuniões curtas

A idéia de reuniões curtas também faz parte dos princípios da XP. Este princípio prega que as reuniões devem ser extremamente objetivas e rápidas. Para garantir

que as reuniões serão breve se objetivas, a XP usa as reuniões em pé. Como não são permitidas cadeiras, a reunião necessariamente tende a ser rápida. Uma boa hora para a realização das reuniões é todos os dias pela manhã, após a chegada da equipe. Forma-se uma roda, onde cada um dos membros faz uma breve atualização do status: o que eles fizeram ontem, o que eles farão hoje e todas as questões ou anúncios dos quais a equipe deve tomar conhecimento.

2.4.3.1.5 Teste contínuo

O teste é outro princípio fundamental da XP. A idéia sobre testes, na XP é testar primeiro e implementar depois. O desenvolvedor deve escrever os testes de unidade, que avalia se uma pequena parte da funcionalidade funciona como o esperado. Esta abordagem é bastante significante para o desenvolvimento, já que quando se testa primeiro, se tem exata noção do comportamento esperado. O código produzido deve passar em todos os testes com 100% de aprovação. Isso permite integrar com confiança e saber que, se um teste falhar durante a integração, o módulo recém-incluído é o que apresenta o erro. Os testes devem ser executados quantas vezes forem possíveis.

2.4.3.1.6 Simplicidade

O conceito de simplicidade é crucial na XP. Sendo extrema, a XP mantém o projeto o mais simples possível. O desenvolvedor deve fazer pequenos projetos para o momento, deve fazer a coisa mais simples que possa funcionar e, deve, também, simplificar o projeto continuamente.

2.4.3.1.7 Programação em Pares

A programação em pares, que segue o conceito de que *“duas cabeças pensam melhor do que uma”*, é empregada na XP. Cada código de linha é desenvolvido por duas pessoas. Um dos desenvolvedores tem o controle do computador e o outro

observa e ajuda. O membro da equipe que observa deve possuir uma noção mais ampla, enquanto que o membro que está escrevendo o código deve se concentrar no problema atual. Isto possibilita que o desenvolvedor que observa possa identificar quando algo está errado. Neste ponto, quando o desenvolvedor observador tem a exata noção do que está acontecendo, ele pode tomar o lugar do outro desenvolvedor, que passa por sua vez a ser o observador do desenvolvimento.

2.4.3.1.8 Padrão de Codificação

A XP emprega ainda um princípio que dita que todos os membros da equipe técnica de desenvolvimento devem adotar e utilizar o mesmo padrão de codificação. O formato deste padrão não importa, desde que seja conhecido e utilizado com consistência por todos os integrantes da equipe. Quando se adota um padrão de codificação, obtem-se várias vantagens como a facilidade de programação em pares, já que o padrão de codificação é o mesmo. Outra importante vantagem obtida com a padronização dos códigos é a velocidade de programação, já que os membros não perdem tempo em identificar padrões diferentes, nem em reformatar o código.

2.4.3.1.9 Propriedade coletiva sobre o código-fonte

Em outros processos de desenvolvimento de software, a propriedade sobre classes é delegada a um dos membros da equipe. Esta situação traz uma série de problemas. Quando, por exemplo, um desenvolvedor precisa de alguma alteração em alguma parte de uma classe que não seja de sua propriedade, ele deve solicitar ao proprietário que faça a alteração. Neste caso, ele deve esperar até que o proprietário tenha tempo para efetuar a alteração. Como nem sempre esta é uma operação rápida, o desenvolvedor que não é o proprietário pode fazer a alteração necessária, gerando uma versão não-oficial daquela classe alterada, o que gera um problema grave. Além disso, esta forma de trabalho cria membros altamente especializados em determinada parte da aplicação, e com conhecimento geral deficiente. Para contornar todos este problemas, a XP apresenta o princípio de

propriedade coletiva. Nesta forma de pensamento, todos os desenvolvedores tem a propriedade sobre todos os módulos do sistema. Sendo assim, quando for necessária uma alteração, a pessoa com melhores condições poderá fazê-la. Cada desenvolvedor deve poder modificar tudo a qualquer momento. Para que esta tarefa não se torne nociva ao processo de desenvolvimento, deve ser empregada uma ferramenta automatizada de gerenciamento de codificação concorrente que esteja disponível.

2.4.3.1.10 Integração contínua

O princípio de integração contínua é um outro pilar da XP. Este princípio dita que todo o código desenvolvido, depois de passar pelo teste de unidade, deve ser integrado ao código principal do sistema. Neste momento, o desenvolvedor tem condições de efetuar testes de integração. Caso o teste falhe, o problema está no novo código que foi inserido e não no código base. Isto facilita a busca e a resolução do problema.

2.4.3.1.11 Refatoração contínua

Outro princípio da XP é efetuar a refatoração todo o tempo. Segundo Fowler (1999), “refatoração é o processo de alterar um sistema de software de tal forma que ele não altere o comportamento externo do código e melhore a estrutura interna. Essa é uma forma disciplinada de limpar o código que minimiza as chances de introdução de bugs”. Sempre que possível, o desenvolvedor deve efetuar a refatoração, para melhorar a qualidade do software como um todo.

2.4.3.1.12 Concepção de pequenas versões

Outra prática empregada pela XP é a concepção de pequenas versões. Com o emprego destas pequenas versões, os usuários recebem as novas funcionalidades o mais rápido possível, o que fornece um feedback valioso sobre o desenvolvimento

estar no caminho certo, além de permitir a detecção precoce de falhas. Outra vantagem desta prática é que, com o emprego de pequenas versões, o desenvolvedor tem a possibilidade de planejar com mais precisão, já que o escopo desta pequena versão é reduzido. No entanto, uma versão só deve ser liberada quando estiver completamente acabada.

2.4.3.1.13 Jornada de Trabalho

A questão de desgaste da equipe também é tratada na XP. Uma equipe de trabalho não deve trabalhar mais tempo do que o necessário, já que isto pode causar estafa. Esta estafa é extremamente prejudicial para qualquer processo de desenvolvimento. Além da carga de trabalho adequada, as pessoas envolvidas no ambiente de trabalho devem incentivar uma quantidade razoável de descontração no trabalho. Todos estes fatores fazem com que o estresse diminua e que o trabalho flua de forma tranquila.

2.4.4 PSP

Segundo Pressman (2006), o PSP (Personal Software Process) é um conjunto estruturado de descrições, medições e métodos de processo, que ajudam os engenheiros a aperfeiçoar seu desempenho pessoal. Fornece os formulários, textos e padrões que os ajudam a estimar e planejar seu trabalho. Mostra-lhes como definir processos e como medir sua qualidade e produtividade.

O PSP baseia-se nos conceitos clássicos de qualidade. E entre os conceitos mais importantes de qualidade encontra-se o de garantia da qualidade, como oposto ao controle de qualidade. Enquanto o controle de qualidade procura encontrar defeitos no produto acabado, a garantia da qualidade procura garantir que, em cada etapa da fabricação do produto, defeitos não sejam injetados (Belloquim, 1999).

Humphrey, partindo da idéia de que para aumentar o nível de maturidade do processo de desenvolvimento como um todo era necessário melhorar a prática dos processos em nível dos desenvolvedores individuais, propôs o PSP (Personal Software Process) (Sommerville, 2003). O PSP tem o objetivo de auxiliar desenvolvedores e pequenas equipes de desenvolvimento em suas crescentes necessidades de desenvolvimento e melhoria de sistemas.

Com a possibilidade de customização do PSP, o engenheiro define quais os elementos que são mais adequados ao seu desenvolvimento. Como no PSP são registrados os resultados obtidos em outros projetos já desenvolvidos, o engenheiro tem a possibilidade de analisar os resultados do seu trabalho passado, com a aplicação do PSP. Isso permite que o processo seja customizado de acordo com suas necessidades, para que os próximos projetos sejam melhorados. Com isso o engenheiro pode gerenciar a qualidade do seu trabalho.
Os objetivos principais do PSP são melhorar o planejamento e o acompanhamento de cronogramas, criar um comprometimento pessoal com a qualidade e um envolvimento constante do desenvolvedor na melhoria contínua do processo. Em outras palavras, o PSP visa permitir que o engenheiro produza software com um nível superior de qualidade, dentro de prazos realisticamente estipulados e com baixos custos de implementação.

O engenheiro que opta pela utilização do PSP tem à sua disposição uma gama de formulários que funcionam como um guia para a coleta e totalização dos dados. Existem formulários específicos para cada parte do projeto: planejamento, desenvolvimento e relatórios de acompanhamento. Tais formulários auxiliam o desenvolvedor a gerenciar de maneira mais efetiva o seu tempo e suas atividades e, também, evidenciam ao desenvolvedor onde o mesmo está cometendo mais erros e se o seu planejamento foi preciso (Sommerville, 2003).

Segundo Sommerville (2003), um projeto que utiliza o PSP como base, passa por sete níveis de maturidade, sendo que cada um destes níveis possuem um objetivo particular. Os níveis de maturidade são agrupados de dois em dois, com excessão do último. Os níveis de maturidade são apresentados e discutidos abaixo.

- **PSP0 e PSP0.1 – O Processo Base**

  O primeiro passo é estabelecer métricas e relatórios padronizados. Estas métricas e relatórios permitem uma base consistente para mensurar o progresso. Tempo, defeito e tamanho são as métricas básicas com as quais o PSP trabalha.

- **PSP1 e PSP1.1 – O Processo de Planejamento Pessoal**

  Este segundo grupo procura ensinar o desenvolvedor a planejar melhor seus projetos. Este nível dedica-se a estabelecer uma forma de estimar o tamanho da atividade, além de se preocupar em introduzir um formato padrão para registrar os dados de teste. Já o PSP1.1 vem acompanhado de um formulário para planejar as tarefas e outro para prever e acompanhar o cronograma de execução. Seu foco é efetuar o planejamento propriamente dito.

- **PSP2 e PSP2.1 – Gerenciamento Pessoal da Qualidade**

  Este nível trata do gerenciamento da qualidade pessoal. Ele ajuda os engenheiros a identificar defeitos precocemente em seus processos. No PSP2, são introduzidas as revisões de código e projeto ao processo do desenvolvedor. O engenheiro deve analisar os defeitos identificados, registrá-los e usar estes dados para estabelecer um *check-list,* que irá orientar a revisão de projetos de desenvolvimento de programas.

- **PSP3 – Processo Pessoal Cíclico**

  Por último, o nível Cyclic Personal Process é oferecido com o objetivo de escalonar projetos maiores. Ele é o último nível do PSP e auxilia o desenvolvedor a lidar com programas maiores. Para isso, divide-se o projeto em partes menores e aplica-se o processo PSP2.1 para cada uma delas. Esses pequenos projetos ficam mais gerenciáveis para o desenvolvedor do que o todo.

A Figura 2.25 mostra os níveis de maturidade do PSP.

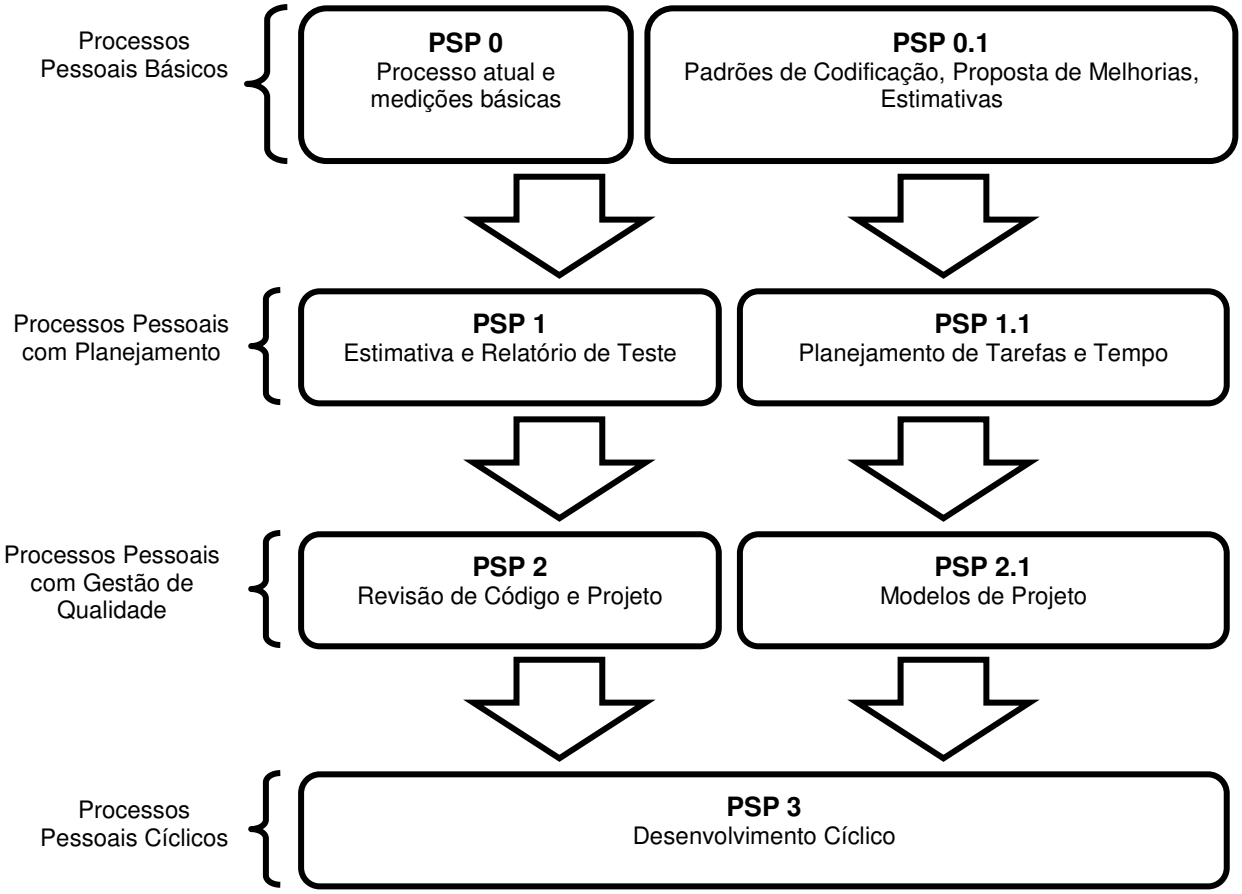

FIGURA 2.25– Fases do PSP (Adaptado de SOMMERVILLE, 2003).

2.4.5 Praxis

A sigla Praxis significa Processo para Aplicativos eXtensíveis InterativoS, refletindo uma ênfase no desenvolvimetno de aplicativos gráficos interativos, baseados na tecnologia orientada a objetos. O Praxis é desenhado para suportar projetos realizados individualmente ou por pequenas equipes, com duração de seis meses a um ano. Com isso, pretende-se que ele seja utilizável para projetos de fim de curso de graduação ou similares, ou projetos de aplicação de disciplinas de engenharia de software. O Praxis abrange tanto métodos técnicos, como requisitos, análise, desenho, testes e implementação, quanto métodos gerenciais, como gestão de requisitos, gestão de projetos, garantia de qualidade e gestão de configurações (PAULA, 2007).

O ciclo de vida do processo é composto por fases que produzem um conjunto precisamente definido de artefatos (documentos e modelos). Para construir cada um destes artefatos, o usuário do processo precisa exercitar um conjunto de práticas recomendáveis da engenharia de software. Na construção destes artefatos, o usuário do processo é guiado por padrões e auxiliado pelos modelos de documentos e exemplos constantes do material de apoio (PAULA, 2003).

O Praxis é baseado na tecnologia orientada a objetos; sua notação de análise e desenho é a UML. Os fluxos do Praxis cobrem as áreas chaves de processos do SW-CMM 1.1. O SW-CMM 1.1 é um modelo de capacitação em processo de software, também adotado por centenas das principais organizações produtoras de software. Ele serve como especificação dos requisitos que devem ser atendidos pelos processos de uma organização, para que essa possa ser considerada madura em desenvolvimento de software. Os padrões do Praxis são conformes com os padrões de engenharia de software do IEEE (1993), os mais abrangentes e respeitados da área (PAULA, 2007; Wikipedia, 2007).

Muitos elementos do Processo Unificado são refletidos no Praxis. Além do Processo Unificado, outros processos também como o PSP influenciaram no desenho do Praxis.
O processo Praxis é dividido nas seguintes fases:

- **Concepção:** Fase na qual as necessidades dos usuários e os conceitos de aplicação são analisados o suficiente para justificar a especificação de um produto de software, resultando em uma proposta de especificação.

- **Elaboração:** Fase na qual a especificação do produto é detalhada o suficiente para modelar conceitualmente o domínio do problema, validar os requisitos em termos desse modelo conceitual e permitir um planejamento acurado da fase de construção.

- **Construção:** Fase na qual é desenvolvida (desenhada, implementada e testada) uma versão completamente operacional do produto, que atende aos requisitos especificados.

- **Transição:** Fase na qual o produto é colocado à disposição de uma comunidade de usuários para testes finais, treinamento e uso final.

Cada uma das fases apresentada acima é detalhada por Paula (2001). Este detalhamento de cada fase do praxis pode ser visto na Tabela 2.5.

TABELA 2.5 – Detalhamento das fases do PRAXIS. (Fonte: PAULA, 2001).

Fase	Iteração	Descrição
Concepção	Ativação	Levantamento e análise das necessidades dos usuários e conceitos da aplicação, detalhados o suficiente para justificar a especificação do software.
Elaboração	Levantamento de Requisitos	Levantamento detalhado das funções, interfaces e requisitos não funcionais do software.
	Análise dos Requisitos	Modelagem conceitual dos elementos relevantes do domínio do problema e uso do modelo para validação dos requisitos e planejamento da fase de construção.
Construção	Desenho Inicial	Definição interna e externa dos componentes do software, em nível suficiente para decidir as principais questões de arquitetura e tecnologia, e para permitir o planejamento das atividades de implementação.
	Liberação 1	Implementação 1 de um subconjunto de funções do produto que será avaliado pelos usuários.
	Liberação N	Implementação N de um subconjunto de funções do produto que será avaliado pelos usuários.
	Liberação Final	Implementação final de um subconjunto de funções do produto que será avaliado pelos usuários.
	Testes Alfa	Realização dos testes de aceitação, no ambiente dos desenvolvedores, juntamente com elaboração da documentação de usuário.
Transição	Testes Beta	Realização dos testes de aceitação, no ambiente dos usuários.
	Operação Piloto	Operação experimental do produto em instalação piloto do cliente, com a resolução de eventuais problemas através de processo de manutenção.

Os fluxos técnicos básicos são os mesmo do Processo Unificado: engenharia de requisitos, análise, desenho, implementação e testes (PAULA, 2007).

Os fluxos técnicos do Praxis são apresentados na Tabela 2.6, bem como sua respectiva descrição resumida.

TABELA 2.6 – Fluxos técnicos do PRAXIS. (Fonte: PAULA, 2001).

Fluxo	Descrição
Requisitos	Fluxo que visa a obter um conjunto de requisitos de um produto, acordado entre cliente e fornecedor.
Análise	Fluxo que visa detalhar, estruturar e validar os requisitos de um produto, em termos de um modelo conceitual do problema, de forma que eles possam ser usados como base para o planejamento e controle detalhados do respectivo projeto de desenvolvimento.
Desenho	Fluxo que visa a formular um modelo estrutural do produto que sirva de base para a implementação, definindo os componentes a desenvolver e a reutilizar, assim como as interfaces entre si e com o contexto do produto.
Implementação	Fluxo que visa a detalhar e implementar o desenho através de componentes de código e de documentação associada.
Testes	Fluxo que visa a verificar os resultados da implementação, através do planejamento, desenho e realização de baterias de testes.
Engenharia de Sistemas	Fluxo que abrange atividades relativas ao desenvolvimento do sistema no qual o produto de software está contido; por exemplo, modelagem de processos de negócio, implantação, usabilidade e criação de conteúdo.

2.5 Metodologia para a Definição de Processos

Como o foco deste trabalho é o desenvolvimento de um processo de desenvolvimento de software científicos, houve a necessidade de identificar uma metodologia para a construção de tal processo. Humphrey (1995) propõe esta metodologia. A metodologia prevê as seguintes etapas:

1. Determinar as necessidades e as prioridades do novo processo
2. Definir os objetivos e os critérios de qualidade
3. Caracterizar o processo atual
4. Caracterizar o processo desejado
5. Estabelecer uma estratégia de desenvolvimento do processo
6. Definir um processo inicial
7. Validar o processo inicial
8. Melhorar o processo

As etapas desta metodologia não precisam ser executadas exatamente nesta seqüência. Mais detalhes sobre a metodologia podem ser vistos em Humphrey (1995).

O próximo capítulo discute os pressupostos para a concepção do processo inicial do PESC.

3. Pressupostos para a Concepção do Processo Inicial

O problema central desta pesquisa pode ser resumido na seguinte questão: como construir software científico de qualidade em pesquisas científicas de forma eficiente e padronizada?

Com o apoio da literatura, uma definição inicial para o PESC foi concebida para responder e dar suporte a este questionamento.

Este capítulo apresenta uma descrição detalhada da metodologia empregada na definição inicial do PESC, bem como sua fundamentação teórica.

Vale aqui registrar o conceito de software científico, gerado por Purri (2006), ao qual o PESC se aplicará:

> "O software científico focado neste trabalho é caracterizado como software de natureza acadêmica desenvolvido por pesquisadores em seus projetos de pesquisa científica. São softwares criados para auxiliar projetos de pesquisa de iniciação científica, mestrado, doutorado, pós doutorado, dentre outros (Purri, 2006).

3.1 Descrição da Metodologia de Desenvolvimento

No desenvolvimento desta pesquisa foram realizadas as seguintes etapas:

- Revisão bibliográfica com ênfase nos tópicos: Engenharia de Software, conceituação do software científico dentro do contexto deste trabalho, hipóteses e diretivas formuladas para a concepção do PESC, processos de desenvolvimento de software, paradigma orientado a objetos e a notação UML.
- Estudo da metodologia proposta por Humphrey (1995) para a definição de um processo.
- Estudo dos processos de desenvolvimento existentes, com ênfase no Processo Unificado e na Extreme Programming.
- Definição de um processo inicial para o PESC, com base nas hipóteses verificadas e diretivas indicadas por Purri (2006).

3.2 Hipóteses Formuladas

Purri (2006), em sua dissertação de mestrado, formulou e verificou a validade de 12 hipóteses acerca do desenvolvimento de software que é praticado atualmente no meio acadêmico. Para tanto, foi realizada uma pesquisa semi-qualitativa, através da implementação de um questionário web (disponibilizado na internet). O perfil das pessoas indicadas para responder o questionário foi de pesquisadores que desenvolvem ou já desenvolveram software científico para suas pesquisas científicas. Para identificar essas pessoas foi utilizada uma consulta na base de dados dos currículos Lattes/CNPq. Foram coletados 1700 e-mails, para os quais a pesquisa foi encaminhada. Deste montante, 501 pesquisadores se interessaram e responderam o questionário. No entanto, 22 questionários foram desconsiderados por não terem sido finalizados, restando 479 questionários que foram considerados aptos para a análise dos resultados. Nesta pesquisa, pesquisadores das mais diversas áreas caracterizaram como desenvolvem seus software e o que julgam importante em seu desenvolvimento (Purri, 2006). Tal questionário foi fundamental para que as atividades propostas por Humphrey fossem realizadas. Isto porque as respostas obtidas serviram como base de dados para a execução das etapas 1, 2 e 3 da Metodologia de Humphrey e auxiliaram na descoberta dos requisitos e características do processo a ser construído.

As hipóteses geradas por Purri (2006) são descritas a seguir e a verificação dos mesmos é apresentada na seção seguinte. Cabe aqui ressaltar que a validação do processo deverá ser feita em uma etapa posterior da pesquisa.

Hipótese 1: A maioria dos pesquisadores não utiliza um processo padronizado para o desenvolvimento de software científico.

As hipóteses de 2 a 4 estão relacionadas às pessoas envolvidas no desenvolvimento de software científico.

Hipótese 2: A equipe envolvida no processo de desenvolvimento de um software científico é pequena, em geral não ultrapassando a três pessoas.

Hipótese 3: Se a grande área de atuação do pesquisador é diferente de ciências exatas e de engenharias, o tamanho da equipe de desenvolvimento do software científico tende a aumentar.

Hipótese 4: As pessoas envolvidas na construção de um software científico possuem bastante conhecimento da área de atuação do software a ser desenvolvido.

As hipóteses de 5 a 12 estão relacionadas às práticas de Engenharia de Software.

Hipótese 5: A comunidade científica procura utilizar técnicas de Engenharia de Software durante o processo de desenvolvimento de software científico, mesmo que de maneira informal e não sistemática.

Hipótese 6: A comunidade científica possui a prática de documentar o processo de desenvolvimento de software científico.

Hipótese 7: O pesquisador que participa da construção do software, além de definir a sua funcionalidade e ser seu principal usuário final, também é um dos maiores produtores dos artefatos gerados no processo de desenvolvimento do software científico.

Hipótese 8: Além dos próprios pesquisadores do projeto, outro grande consumidor dos artefatos produzidos durante o processo de desenvolvimento de software científico são pesquisadores que utilizarão o software construído em outros projetos de pesquisa.

Hipótese 9: O orientador de um projeto de pesquisa que necessita da construção de um software científico é um pequeno produtor de artefatos do processo de desenvolvimento deste software.

Hipótese 10: Os pesquisadores das grandes áreas de ciência exatas e de engenharia dão maior importância a utilização de alguma modelagem do processo de desenvolvimento do software científico do que as demais áreas.

Hipótese 11: O processo de desenvolvimento de software científico deve ter uma preocupação especial com a atividade de integração.

Hipótese 12: Hoje, no desenvolvimento de software científico, pouca atenção é dada ao estudo dos riscos do projeto.

3.3 Análise das Respostas e Verificação das Hipóteses

A primeira hipótese formulada foi de que a maioria dos pesquisadores não utiliza um processo padronizado para o desenvolvimento de software científico. Percebeu-se na análise das respostas que a maioria dos entrevistados utilizam práticas de Engenharia de Software na construção de software científico e não um processo bem definido propriamente dito. Sendo assim, esta hipótese foi verificada, fato este que gerou forte motivação para a concepção do PESC. Cabe ressaltar que o Processo Unificado (PU) foi muito citado entre os entrevistados que afirmaram utilizar um processo. Com base neste resultado, o PESC foi fundamentado neste processo, com algumas customizações advindas das outras hipóteses formuladas.

Já na Hipótese 2, foi verificado que a equipe de desenvolvimento do software científico é restrita em número de pesquisadores, variando de 1 a 3 pessoas. Sendo assim, o PESC foi desenvolvido para equipes pequenas

Na terceira hipótese formulada, acreditava-se que o tamanho da equipe de desenvolvimento de software científico tenderia a aumentar quando os trabalhos de pesquisa em desenvolvimento não fossem da grande área de atuação das ciências exatas e engenharia. Como a gama de respostas do

grupo pertencente à outras grandes áreas de pesquisa foi reduzida, esta hipótese não pôde ser verificada.

Ainda com base nestas respostas, não foi possível comparar o tamanho da equipe de desenvolvimento atuante nas áreas de ciências exatas e engenharias com as demais áreas do conhecimento, pelo mesmo motivo acima. Além disso, fatores como o tamanho e a complexidade do projeto e o conhecimento da área de aplicação podem fazer com que o tamanho da equipe varie consideravelmente.

Na Hipótese 4 buscava-se verificar se as pessoas envolvidas na construção de um software científico possuem bastante conhecimento da área de atuação do software a ser desenvolvido. Esta hipótese foi verificada com ressalvas, pois mensurar o grau de conhecimento de uma pessoa sobre determinado assunto é bastante subjetivo. No entanto, os resultados obtidos em algumas questões confirmaram a hipótese de que as pessoas envolvidas na construção de um software científico possuem bastante conhecimento da área de atuação do software a ser desenvolvido.

A Hipótese 5, que buscava verificar se a comunidade científica procura utilizar técnicas de Engenharia de Software durante o processo de desenvolvimento de software científico, mesmo que de maneira informal e não sistemática, pôde ser verificada, já que a análise das respostas do questionário indicam uma tendência dos entrevistados em utilizar técnicas de Engenharia de Software no desenvolvimento de software científico. Foi verificado que parte dos pesquisadores que afirmaram utilizar um processo padronizado no desenvolvimento de software científico na verdade aplicam práticas de Engenharia de Software. Outras importantes atividades da Engenharia de Software como, por exemplo: definição de requisitos, construção do diagrama de classes, plano de testes, projeto arquitetônico, dentre outras, são executadas por grande parte dos entrevistados no desenvolvimento de software científico e muitas dessas atividades são executadas de maneira informal. Os entrevistados indicaram ainda que utilizavam algum tipo de documentação no desenvolvimento de software científico. Dentre as

informações citadas como parte integrante destes documentos, estão técnicas canônicas da Engenharia de Software, como, por exemplo: descrição de casos de uso, restrições de projeto e plano de testes.

A premissa de que a comunidade científica possui a prática de documentar o processo de desenvolvimento de software científico e que era o tema da Hipótese 6, pôde ser verificada. Informações acerca do software científico, como funcionalidades gerais, descrição dos casos de uso e restrições do projeto foram largamente citadas pelos entrevistados, o que indica uma forte preocupação em se documentar o processo de desenvolvimento do software científico em questão. Outro ponto muito citado nas respostas foram os testes. Levando em consideração a natureza complexa do software científico, era de se esperar esta tendência, já que os resultados gerados por estes software serão importantes para a validação de pesquisas acadêmicas na qual estão inseridos.

As hipóteses 7 e 8 diziam respeito ao gerador de artefatos e funcionalidades do software científico, bem como de seu consumidor final. Esperava-se nestas hipóteses que o pesquisador fosse o principal responsável pelo desenvolvimento do software científico como um todo e também seu principal consumidor final. Além deste, outros pesquisadores poderiam também ser grandes consumidores do software científico construído, aproveitando-o como um todo ou parcialmente na continuação da pesquisa em questão ou no desenvolvimento de outras pesquisas. As hipóteses 7 e 8 foram verificadas, como era esperado. Em suas respostas, os entrevistados afirmaram que os maiores produtores de artefatos do processo de desenvolvimento de software científico são os próprios pesquisadores, seguidos dos analistas e dos programadores do software, e os maiores consumidores são os pesquisadores que utilizarão o software em outros projetos de pesquisas e os pesquisadores envolvidos no projeto de pesquisa. Acreditava-se ainda que o orientador de um projeto de pesquisa que necessita da construção de um software científico é um pequeno produtor de artefatos do processo de desenvolvimento deste software, tema da Hipótese 9. No entanto esta hipótese não pôde ser

verificada, já que um número considerável de entrevistados (cerca de 41%) afirmou que o orientador é um produtor de artefatos.

Na décima hipótese, buscava-se verificar se os pesquisadores das grandes áreas de ciências exatas e de engenharias dão maior importância à utilização de alguma modelagem do processo de desenvolvimento do software científico do que as demais áreas. Assim como na Hipótese 3, esta hipótese também não pôde ser verificada já que a quantidade de respostas pertencentes às outras grandes áreas do conhecimento foi reduzida, fato este que não permitiu uma conclusão que retratasse o desenvolvimento de software científico cotidiano dos entrevistados. Porém, o que pôde ser verificado é que os entrevistados da área de ciência da computação demonstraram bastante interesse na utilização da linguagem UML na modelagem do processo.

A Hipótese 11 que visava verificar se o processo de desenvolvimento de software científico deve ter uma preocupação especial com a atividade de integração e reuso foi verificada. A integração foi classificada com prioridade alta média por cerca de 40% dos entrevistados, seguido de 34% como prioridade média, o que indica que esta é uma grande necessidade dos desenvolvedores de software científico. A atividade de integração está ligada diretamente ao reuso de componentes. Fatores como a definição da interface de integração, a realização de testes de integração, bem como a documentação da integração para facilitar a legibilidade devem ser levados em consideração.

Outro ponto importante é o fator *reuso,* que foi fortemente indicado como primordial pela maioria dos entrevistados. Com base nos resultados analisados, percebe-se claramente a importância do reuso de componentes no processo de desenvolvimento de software científico. Conseqüentemente, os resultados sugerem também uma atenção especial com a atividade de integração no processo de desenvolvimento de software científico, já que as atividades relacionadas a integração e ao reuso estão intrinsecamente ligadas.

Na décima segunda e última hipótese, buscava-se verificar se atualmente, no desenvolvimento de software científico, pouca atenção é dada ao estudo dos riscos do projeto. Esta hipótese foi verificada já que apenas cerca de 24% dos entrevistados demonstraram a preocupação em documentar os riscos do projeto.

3.4 Diretrizes Iniciais para o PESC

A partir das análises realizadas sobre os dados obtidos dos questionários respondidos, da avaliação das respostas relativas às hipóteses formuladas e da aplicação das três primeiras etapas da Metodologia de Humphrey, algumas diretrizes para a definição do PESC foram propostas em Purri (2006).

Vale salientar que o processo inicial do PESC é direcionado para produzir software científico de natureza acadêmica e relacionado às áreas de ciências exatas e de engenharias, já que a maior parte dos entrevistados que geraram a base de conhecimento para a pesquisa pertencem a estas áreas (Pereira Jr *et .al.*, 2007).

Abaixo é apresentada uma compilação das características que o processo inicial do PESC deveria possuir, segundo as diretrizes propostas e encontradas em Purri (2006). São elas:

1. Ciclo iterativo e incremental;
2. Base no Processo Unificado;
3. Deve ser um processo simples;
4. Deve ser voltado para uma equipe pequena de desenvolvimento;
5. Deve utilizar a linguagem UML como base;
6. Deve permitir o gerenciamento de código aberto e concorrente;
7. A utilização dos artefatos deve ser opcional.

Assim como o processo apresenta algumas características essenciais, os artefatos que o comporão também deverão possuir algumas peculiaridades. Segundo Purri (2006), as características são as seguintes:

1. Descrição das funcionalidades gerais do software;
2. Projeto arquitetural;
3. Definição do tipo de licença do software;
4. Descrição da relevância científica do software;
5. Documentação do código-fonte e regras de codificação;
6. Descrição das restrições e viabilidade do projeto;
7. Levantamento completo e especificação detalhada dos requisitos;
8. Diagramas de casos de uso, descrição e especificação;
9. Diagramas de classes completo;
10. Citação das referências científicas nas quais a implementação irá se basear;
11. Descrição dos algoritmos complexos;
12. Documentação dos componentes reutilizados na construção do software;
13. Documentação de módulos lógicos reutilizáveis em outros projetos;
14. Plano e descrição dos testes
15. Relatórios de erros encontrados e causas;
16. Registro das falhas e dos sucessos do projeto de software;
17. Manual do usuário do software.

3.5 Concepção do Processo Inicial

O processo inicial foi concebido com base na observância das características demonstradas na seção anterior, além de ter sofrido forte influência de outros processos de desenvolvimento de software consagrados, principalmente do Processo Unificado (PU) e da Extreme Programming (XP).

As características principais que foram adaptadas do Processo Unificado são o ciclo de vida iterativo e incremental, e a agilidade do processo. Tanto o PESC quanto o processo unificado são processos com poucos artefatos, com pouca

burocracia e que permitem um rápido desenvolvimento de software. No entanto, o PESC tem um enfoque científico, enquanto que o Processo Unificado tem um enfoque comercial.

Além do Processo Unificado, a Extreme Programming (XP) também exerceu forte influência sobre o processo inicial do PESC. A XP forneceu até mais elementos que o próprio Processo Unificado (PU), até então tido como base para o PESC. Isso se deu pelo fato de a XP se assemelhar mais com as características levantadas por Purri (2006) em seu trabalho que o PU. Dentre as características da XP que podem ser encontradas no PESC, nota-se a fase de planejamento em uma das fases das suas iterações, a padronização do código-fonte, a simplicidade do processo, a concepção de pequenas versões em cada iteração do ciclo de vida iterativo e incremental, a integração contínua e a propriedade coletiva sobre o código-fonte.

O processo inicial, com todas as suas características e artefatos, é apresentado no próximo capítulo.

4. Processo Proposto

Este trabalho de pesquisa tem como objetivo apresentar um processo inicial para o PESC. A partir das diretrizes geradas por Purri (2006) em sua dissertação de mestrado, foi possível o desenvolvimento deste processo específico para o desenvolvimento de software científico. Este capítulo caracteriza o processo gerado, apresentando uma discussão sobre o seu ciclo de desenvolvimento e também de seus artefatos.

É de suma importância ressaltar que o PESC é voltado para o desenvolvimento de sistemas orientados a objetos, já que é a abordagem mais moderna de desenvolvimento.

Os artefatos do PESC, no seu formato original, estão disponíveis no Anexo A desta dissertação.

4.1 O processo inicial

O processo inicial do PESC é formado por seis artefatos, sendo cinco deles relacionados à uma área do desenvolvimento e um específico para o controle geral do desenvolvimento do software. Estes artefatos constituem um arcabouço de desenvolvimento, onde o pesquisador/desenvolvedor tem a opção de omitir determinadas informações destes documentos, fato este que faz com que o PESC se torne um processo altamente customizável. No entanto, algumas partes dos artefatos são obrigatórias, visando a documentação mínima necessária para o controle do processo de desenvolvimento. Para permitir uma fácil identificação das partes obrigatórias e opcionais de cada artefato, foi adotada a seguinte notação: os títulos dos artefatos que estão em vermelho são obrigatórios, enquanto que os títulos dos artefatos que estão em verde são opcionais.

Cabe aqui ressaltar que, idealmente, o PESC deve ser um processo automatizado. Sendo assim, deve existir uma ferramenta, preferencialmente web, que implemente o processo. Nesta ferramenta automatizada, o pesquisador selecionará e usará os

artefatos mais adequados ao seu desenvolvimento. Além da seleção dos documentos que serão utilizados para documentar o projeto, a ferramenta deverá controlar as alterações destes documentos. Esta funcionalidade permitirá que todos os desenvolvedores participantes de um determinado projeto acompanhem a sua evolução. No entanto, o foco desta parte da pesquisa foi a definição propriamente dita do processo, para que uma posterior automatização seja possível.
Na seção subsequente é apresentado o ciclo de desenvolvimento do PESC, bem como o detalhamento de cada uma de suas iterações.

4.1.1 Ciclo de Desenvolvimento

O ciclo de desenvolvimento utilizado no PESC é aquele conhecido como iterativo ou incremental, que é empregado no PU (Processo Unificado) e na XP (Extreme Programming). Como o ciclo de vida é evolutivo, ele é composto por iterações. Ao final de cada iteração, tem-se uma parte executável do sistema, conhecida como liberação ou versão. Em cada uma das iterações são contempladas as seguintes fases:

- ***Planejamento***

  Cada uma das iterações deve ser extremamente reduzida. Isto permite um controle muito maior do desenvolvimento, já que se pensa apenas em uma pequena parte do projeto de cada vez. Cada uma das iterações deve ser planejada de modo que seja simples, mas que não deixe de lado nenhum detalhe importante ao desenvolvimento.

- ***Desenvolvimento***

  Com base no planejamento rápido feito anteriormente, é desenvolvida a versão do incremento em questão, observando-se os padrões de codificação. É gerado então uma versão em cada iteração do ciclo de vida.

- ***Testes***

  Após o seu desenvolvimento, cada versão deve passar pelos testes de unidade e integração. Os testes de unidade servem para garantir que esta versão está

funcionando de acordo com o planejamento efetuado. Após a aprovação desta versão no teste de unidade, deve-se proceder com o teste de integração, onde será verificada a compatibilidade deste novo código com o que já existia anteriormente. Esta técnica permite identificar precocemente falhas na integração do sistema e permite saber que, se um teste falhar durante a integração, o módulo recém-incluído é o que apresenta o erro.

- ***Implantação***

  Depois de passar pelos testes de unidade e integração, a nova versão deve ser incorporada ao código-fonte oficial.

A Figura 4.1 mostra o ciclo de desenvolvimento do PESC.

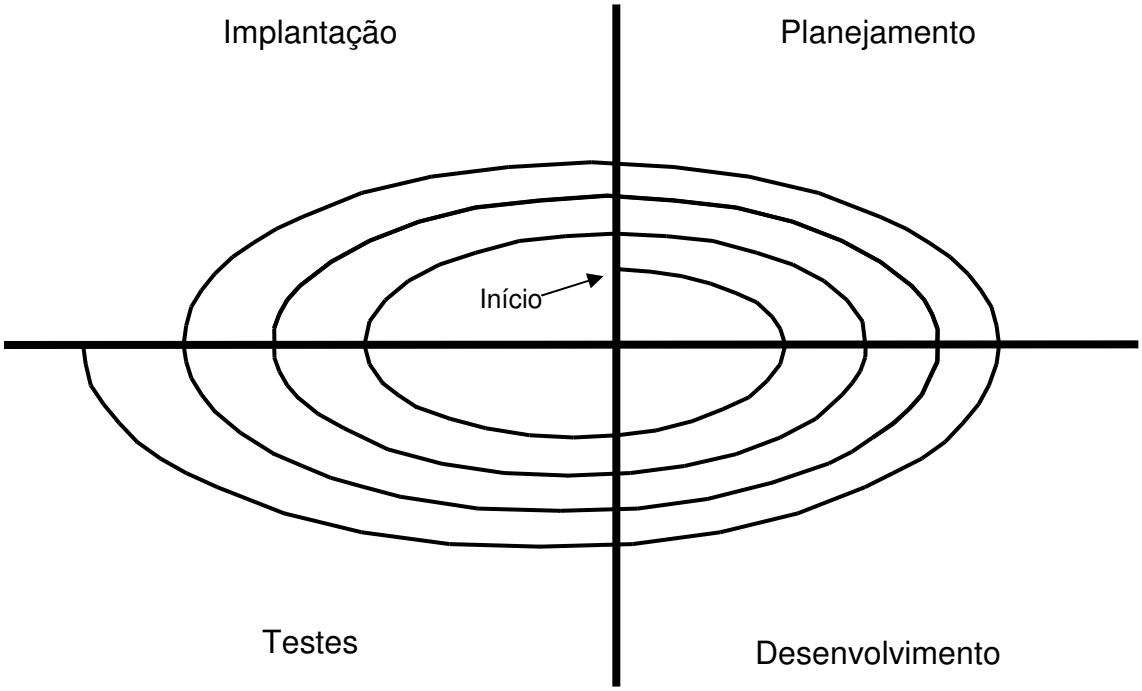

FIGURA 4.1 – Ciclo de Vida do PESC.

Estas iterações devem acontecer até que se tenha o software completo.

Na seção seguinte, são detalhados os artefatos do PESC, bem como a sua aplicação dentro de cada uma das fases do ciclo de desenvolvimento definido anteriormente.

4.1.2 Artefatos

Como padrão, cada um dos artefatos possui, na capa, o nome da instituição de pesquisa, o nome do projeto que está sendo desenvolvido, o número da iteração que está sendo feita, o nome do próprio artefato e, ao final, o mês e o ano de desenvolvimento do projeto. Cada um dos artefatos possui seções auto-explicativas próprias, que auxiliam o desenvolvedor no seu preenchimento.

4.1.2.1 Controle Geral de Desenvolvimento

Este artefato é conhecido como artefato guarda-chuva[1]. Sendo assim, este artefato é específico para o controle geral do desenvolvimento do software. As partes deste documento são mostradas a seguir:

Na primeira seção do documento, o desenvolvedor deve registrar as finalidades e demais informações pertinentes ao desenvolvimento. As informações detalhadas que devem obrigatoriamente ser informadas são:

- **Nome do Software:** nesta seção o desenvolvedor deve registrar o nome do software que será desenvolvido.
- **Integrantes/Desenvolvedores:** nesta seção o desenvolvedor deve registrar o nome dos integrantes da equipe de desenvolvimento do software.
- **Definição da Licença do Software:** nesta seção o desenvolvedor deve registrar o tipo de licença que o software terá.
- **Funcionalidades gerais do Software:** nesta seção o desenvolvedor deve registrar a descrição das funcionalidades gerais do software, informando suas finalidades e demais informações pertinentes ao desenvolvimento.
- **Descrição da Relevância Científica do Software:** nesta seção o desenvolvedor deve registrar a descrição da relevância do software que será construído para a comunidade científica.

[1] Um artefato guarda-chuva é aquele documento que compreende todo o processo de desenvolvimento, ou seja, um documento que é utilizado e/ou modificado em todas as fases do desenvolvimento.

- **Padrão de Codificação:** nesta seção o desenvolvedor deve registrar o padrão de codificação que será empregado durante todo o ciclo de desenvolvimento do software. Neste padrão deve ser descrita a forma como o código-fonte do software será escrito. Sendo assim, o pesquisador registra, por exemplo, o padrão dos nomes que são utilizados nas declarações de variáveis.

A segunda seção permite que o desenvolvedor documente os módulos que eventualmente serão desenvolvidos dentro do sistema. As informações contidas nesta seção, que são obrigatórias caso o projeto seja formado por módulos, são mostradas abaixo:

- **Nome do Módulo:** nesta seção o desenvolvedor deve registrar o nome do módulo que está previsto para ser desenvolvido.
- **Responsável(is):** nesta seção o desenvolvedor deve registrar o nome do(s) responsável(is) pelo desenvolvimento do módulo.
- **Descrição Geral do Módulo:** nesta seção o desenvolvedor deve registrar a descrição geral do módulo, informando suas funcionalidades, finalidades, modelo conceitual e demais informações pertinentes ao desenvolvimento.
- **Início do Desenvolvimento:** nesta seção o desenvolvedor deve registrar a data em que a construção do módulo foi iniciada.

A terceira seção permite que o desenvolvedor documente a arquitetura do projeto. Uma arquitetura geralmente é necessária quando a complexidade de um sistema é muito grande. A arquitetura permite um gerenciamento macro do projeto. Esta seção é opcional e deve ser preenchida somente se o projeto apresentar uma arquitetura própria. As informações contidas nesta seção, que são obrigatórias caso o projeto possua uma arquitetura, são mostradas abaixo:

- **Arquitetura do Projeto:** nesta seção o desenvolvedor deve apresentar a arquitetura do projeto.

- **Descrição:** nesta seção o desenvolvedor deve registrar uma descrição geral sobre a arquitetura do projeto, enfatizando as interações entre seus componentes internos e externos.

Ainda neste artefato o desenvolvedor deve, obrigatoriamente, registrar as alterações que forem efetuadas em cada artefato, para que possa ter um controle do processo como um todo. Estes dados acabam por gerar um histórico do desenvolvimento do sistema. Esta seção apresenta as seguintes informações obrigatórias:

- **Nome do Artefato:** nesta seção o desenvolvedor deve registrar o nome do artefato que foi alterado.
- **Versão do Artefato que sofreu a alteração:** nesta seção o desenvolvedor deve registrar a versão do artefato que foi alterada.
- **Data da Alteração:** nesta seção o desenvolvedor deve registrar a data em que a construção do módulo foi iniciada.
- **Autor da Alteração:** nesta seção o desenvolvedor deve registrar o nome do desenvolvedor que efetuou a alteração.
- **Alterações Efetuadas:** nesta seção o desenvolvedor deve registrar as alterações que foram efetuadas, a fim de criar um histórico de cada versão do documento.

Todas as informações deste documento são obrigatórias, pois através dele é que será feito todo o controle do processo de desenvolvimento.

4.1.2.2 Requisitos e Escopo da Versão

Este artefato é utilizado e/ou modificado na fase de planejamento, que é a primeira etapa do ciclo de desenvolvimento do PESC. Neste artefato, o desenvolvedor deve registrar o que foi planejado para aquela iteração. As informações que compõem a primeira seção deste documento são:

- **Descrição dos Requisitos da Versão:** nesta seção o desenvolvedor deve detalhar todos os requisitos levantados para o desenvolvimento da versão atual.
- **Funcionalidades Esperadas da Versão:** nesta seção o desenvolvedor deve detalhar as funcionalidades que se espera da versão que será construída.
- **Escopo da Versão:** nesta seção o desenvolvedor deve registrar o escopo, ou seja, a área de abrangência da versão que será desenvolvida.
- **Início do Desenvolvimento:** nesta seção o desenvolvedor deve registrar a data em que a construção do módulo foi iniciada.

Estas seções são obrigatórias, pois compõem os dados de documentação mínimos exigidos.

Existem ainda outras informações que são opcionais e podem ser omitidas caso não sejam pertinentes no desenvolvimento de uma determinada versão. Estas informações opcionais estão presentes na segunda seção do documento e são mostradas abaixo:

- **Limitações da Versão:** nesta seção o desenvolvedor deve detalhar as eventuais limitações da versão que será construída.
- **Restrições de Performance:** nesta seção o desenvolvedor deve detalhar as eventuais restrições de performance da versão que será construída.
- **Restrições de Usabilidade, Adaptabilidade e Portabilidade:** nesta seção o desenvolvedor deve detalhar as eventuais restrições de usabilidade, adaptabilidade e portabilidade da versão que será construída.
- **Restrições de Hardware:** nesta seção o desenvolvedor deve detalhar as eventuais restrições de hardware da versão que será construído.
- **Viabilidade:** nesta seção o desenvolvedor deve detalhar um eventual estudo de viabilidade da versão que será construída.

Cada iteração do ciclo de vida deve gerar um documento com todas as informações apresentadas acima.

Após a confecção deste artefato, o desenvolvedor deve se concentrar no artefato de Detalhamento dos Casos de Uso, que é detalhado na próxima seção.

4.1.2.3 Detalhamento dos Casos de Uso

Este artefato é modificado ainda na fase de planejamento. Ele é confeccionado com base nas informações coletadas e documentadas no artefato *Controle Geral de Desenvolvimento.*

Neste artefato, o desenvolvedor conta com a seguinte seção:

- **Diagrama de Casos de Uso:** nesta seção o desenvolvedor deve apresentar o diagrama de casos de uso, além de ter a possibilidade de apresentar uma breve descrição sobre o mesmo, caso seja necessário.

O diagrama de casos de uso é obrigatório, já que representa, de forma gráfica, as funcionalidades mínimas que o sistema deverá implementar.

As próximas informações do documento contemplam o detalhamento do diagrama de casos de uso mostrado na primeira seção do artefato. Estas informações devem ser replicadas para cada caso de uso presente no diagrama. As informações que são obrigatórias são mostradas abaixo:

- **Nome do Caso de Uso:** nesta seção o desenvolvedor deve registrar o nome do caso de uso.
- **Atores:** nesta seção o desenvolvedor deve detalhar os atores e suas relações com o caso de uso.
- **Fluxo de Eventos:** nesta seção o desenvolvedor deve detalhar o fluxo de eventos que ocorre com um determinado caso de uso. Engloba os eventos que estão explicitados no diagrama de Caso de Uso. Esta seção oportunamente pode ser dividida em Fluxo Básico e Fluxos Alternativos. O

Fluxo Básico descreverá a ação feita pelo ator correspondente e a conseqüente resposta do sistema. Os Fluxos Alternativos apresentam alternativas ao Fluxo Básico, podendo representar, por exemplo, exceções que acontecem no Fluxo Básico.

Um caso de uso pode ainda possuir outras informações particulares, que também são contempladas por este artefato, tais como as mostradas a seguir:

- **Requisitos Especiais:** nesta seção o desenvolvedor deve detalhar os eventuais requisitos especiais de um caso de uso.
- **Pré-condição:** nesta seção o desenvolvedor deve detalhar as eventuais pré-condições exigidas por um caso de uso.
- **Pós-condição:** nesta seção o desenvolvedor deve detalhar as eventuais pós-condições geradas por um caso de uso.
- **Pontos de Extensão:** nesta seção o desenvolvedor deve detalhar os eventuais pontos de extensão de um caso de uso.

Como estas informações não são aplicadas à todos os casos de uso, cabe ao desenvolvedor utilizar ou omití-las, tornando-as assim opcionais.

Cada iteração deve, idealmente, produzir um artefato completo de *Detalhamento dos Casos de Uso.*

Após o preenchimento deste artefato, o desenvolvedor terá a base necessária para desenvolver o artefato de *Plano de Codificação da Versão*. Este artefato é apresentado, em detalhes, na seção seguinte.

4.1.2.4 Plano de Codificação da Versão

Este artefato é modificado na fase de planejamento e deve ser consultado e, se necessário, modificado na fase de desenvolvimento. Caso este artefato seja alterado, deve-se registrar o ocorrido no artefato *Controle Geral de Desenvolvimento*. O artefato *Plano de Codificação da Versão* é particularmente

importante para o sucesso do desenvolvimento, pois define todos os detalhes pertinentes ao desenvolvimento do código-fonte propriamente dito, incluindo os diagramas de classes. Este artefato é, fundamentalmente, formado a partir do artefato *Detalhamento dos Casos de Uso.*

A primeira seção deste artefato apresenta as seguintes informações obrigatórias:

- **Nome ou Identificador da Versão:** nesta seção o desenvolvedor deve registrar o nome ou identificador da versão que será construída na iteração corrente.
- **Diagramas de Classes:** nesta seção o desenvolvedor deve apresentar os diagramas de classes, além de ter a possibilidade de apresentar uma breve descrição sobre os mesmos, caso seja necessário.

As próximas informações do documento se referem ao detalhamento do diagrama de classes mostrado na primeira seção deste artefato. Estas informações devem ser replicadas para cada classe presente no diagrama. As informações que são obrigatórias são mostradas abaixo:

- **Nome e Finalidade da Classe:** nesta seção o desenvolvedor deve registrar o nome e a finalidade de cada classe dentro do diagrama de classes.
- **Atributos:** nesta seção o desenvolvedor deve registrar os atributos da classe que está sendo detalhada. Devem ser colocados o nome do atributo, o seu tipo e visibilidade, além de indicar a sua importância perante os requisitos já identificados.
- **Operações:** nesta seção o desenvolvedor deve registrar as operações da classe que está sendo detalhada. Devem ser colocados o nome da operação, seus eventuais parâmetros e valores de retorno, além da sua visibilidade, além de indicar detalhes do seu funcionamento.
- **Relacionamentos:** nesta seção o desenvolvedor deve registrar os relacionamentos da classe que está sendo detalhada.
- **Responsabilidades:** nesta seção o desenvolvedor deve registrar as responsabilidades da classe que está sendo detalhada.

Uma segunda seção deste documento permite a documentação dos componentes reutilizados no desenvolvimento da versão atual. As seguintes informações devem ser preenchidas, caso se tenha componentes reutilizados:

- **Nome ou identificador do Componente:** nesta seção o desenvolvedor deve registrar o nome do componente que está sendo reutilizado.
- **Responsabilidades:** nesta seção o desenvolvedor deve registrar as responsabilidades deste componente que está sendo reutilizado.
- **Interface de Comunicação:** nesta seção o desenvolvedor deve detalhar e registrar a interface do componente que está sendo reutilizado.

Caso não existam componentes reutilizados, a seção acima deve ser omitida do artefato.

A terceira seção do documento, denominada de *Descrição de Algoritmos Complexos*, dá a possibilidade ao desenvolvedor de efetuar a descrição de alguma parte do código-fonte considerada complexa, sempre que for necessário. Esta seção é opcional e deve ser omitida quando não existir a necessidade de explicação detalhada de algoritmos.

O desenvolvedor pode ainda, caso seja necessário, documentar as referências científicas em que se baseia uma determinada implementação, na quarta seção do artefato, que recebe este mesmo nome. Sendo assim, esta seção é opcional e só se aplica aos casos que o pesquisador julgar relevantes.

Este artefato contempla, ainda, uma última seção opcional, que trata da documentação de componentes gerados durante o processo de desenvolvimento. A informação contida nesta seção é detalhada abaixo:

- **Detalhamento da Interface do Componente:** Nesta seção a interface dos componentes gerados deve ser detalhada, indicando os métodos e

parâmetros que permitem a comunicação de dados entre esse componente e o sistema como um todo.

Neste artefato, o desenvolvedor tem algumas seções opcionais e a sua utilização dependerá da análise da sua necessidade pelo pesquisador.

Cada iteração deve, idealmente, produzir um artefato completo do *Plano de Codificação da Versão*.

Após determinar o *Plano de Codificação da Versão*, o pesquisador deve então produzir o código-fonte propriamente dito, com base nas informações contidas neste artefato e observando o padrão de codificação estabelecido no artefato *Controle Geral de Desenvolvimento*. Ao final da codificação, o pesquisador tem as informações necessárias para passar para a próxima etapa do ciclo de desenvolvimento, que é a fase de testes. Esta fase é documentada com o artefato *Plano de Testes*, que é mostrado a seguir.

4.1.2.5 Plano de Testes

Este artefato é utilizado na fase de testes. O código-fonte gerado na fase anterior do ciclo de vida servirá de base para a aplicação dos testes que serão documentados neste artefato.

Neste artefato, o desenvolvedor tem a possibilidade de documentar todo o processo de testes que é executado. Como descrito no ciclo de vida do PESC, devem ser efetuados testes de unidade e posteriormente testes de integração. Os testes de unidade permitem que o desenvolvedor garanta que a versão construída está funcionando de acordo com o planejamento efetuado na fase anterior. Após passar pelo teste de unidade, a versão deve ser submetida à testes de integração, onde será verificada a compatibilidade deste novo código com o que já existia anteriormente. Esta técnica permite identificar precocemente falhas na integração do sistema e permite saber que, se um teste falhar durante a integração, o módulo recém-incluído é o que apresenta o erro, preservando assim o código-fonte principal.

As partes do artefato que tratam dos testes de unidade são: Testes de Código-fonte e Testes de Interface. Na seção de testes de código-fonte, devem ser aplicadas e documentadas técnicas de Engenharia de Software que são conhecidas como Teste de Caixa-Branca. Este tipo de teste tem o foco principal no código-fonte, deixando de lado a interface da versão. Esta seção apresenta algumas informações obrigatórias que são:

- **Objetivos do teste:** Esta seção do artefato deve registrar os objetivos do teste que será aplicado.
- **Técnica de Teste:** nesta seção, deve ser indicada a técnica de teste que será empregada, já que existem distintas formas de se proceder com um teste de caixa-branca. Cabe ao desenvolvedor selecionar o tipo de técnica que mais seja aplicável ao seu desenvolvimento e aplicá-la.
- **Critérios de Sucesso:** nesta seção, devem ser descritos os critérios que devem ser alcançados para que o teste seja considerado um sucesso.
- **Resultados Obtidos:** os resultados obtidos em cada teste devem ser registrados nesta seção para comparação com os critérios de sucesso da seção anterior.

Já na parte de testes de interface, devem ser aplicadas e documentadas técnicas de Engenharia de Software que são conhecidas como Teste de Caixa-Preta. Este tipo de teste tem o foco principal na interface da versão, deixando de lado o seu código-fonte. Esta parte do documento apresenta algumas informações obrigatórias que são:

- **Objetivos do teste:** Esta seção do artefato deve registrar os objetivos do teste que será aplicado.
- **Técnica de Teste:** nesta seção, deve ser indicada a técnica de teste que será empregada, já que existem distintas formas de se proceder com um teste de caixa-preta. Cabe ao desenvolvedor selecionar o tipo de técnica que mais seja aplicável ao seu desenvolvimento e aplicá-la.

- **Dados de Entrada:** nesta seção, devem ser detalhados os dados de entrada que serão utilizados no teste. Esta informação é importante pois permite que outro desenvolvedor da equipe efetue o teste com os mesmos dados de entrada.
- **Dados esperados de saída (Critérios de Sucesso):** nesta seção, devem ser descritos os dados que são esperados como saída do processamento. Estes dados devem ser confrontados com os dados da próxima seção.
- **Resultados Obtidos:** esta seção deve registrar os resultados obtidos em cada teste. Estes resultados devem ser comparados com os dados esperados de saída, para determinar se o teste foi concluído com sucesso.

Já a seção do artefato que permite a documentação dos testes de integração recebe exatamente este nome. Esta seção apresenta as seguintes informações obrigatórias:

- **Objetivos do teste:** Esta seção do artefato deve registrar os objetivos do teste que será aplicado.
- **Técnica de Teste:** nesta seção, deve ser indicada a técnica de teste que será empregada. Cabe ao desenvolvedor selecionar o tipo de técnica que mais seja aplicável ao seu desenvolvimento e aplicá-la.
- **Critérios de Sucesso:** nesta seção, devem ser descritos os critérios que devem ser alcançados para que o teste seja considerado um sucesso.
- **Resultados Obtidos:** os resultados obtidos em cada teste devem ser registrados nesta seção, para comparação com os critérios de sucesso, da seção anterior.

O artefato apresenta ainda algumas seções que eventualmente podem ser utilizadas. Uma delas permite que sejam documentados testes de performance, caso sejam aplicáveis. Esta seção deve ser omitida do documento, caso não seja aplicável ao desenvolvimento atual. Outra seção presente no artefato, a de Testes de Validação de Requisitos é particularmente importante, pois dá suporte ao desenvolvimento. Nesta seção, o desenvolvedor tem a possibilidade de confrontar os requisitos identificados na fase de planejamento com as funcionalidades realmente implementadas.

Cada iteração do ciclo de vida do PESC deve gerar um artefato de testes. Após a realização de todos os testes, o desenvolvedor tem ainda mais um artefato que permite o registro das falhas e sucessos do desenvolvimento. Após a execução dos testes e conseqüente documentação registrada no artefato *Plano de Testes*, o pesquisador deve integrar a nova versão do código produzida na versão oficial do sistema, terminando assim uma iteração do ciclo de desenvolvimento. Além disso, de posse das informações contidas no documento de *Plano de Testes,* o pesquisador tem condições de registrar as informações contempladas por um último artefato do PESC, o de *Registro de Falhas e Sucessos*, que é descrito a seguir.

4.1.2.6 Registro das Falhas e Sucessos

Este artefato é utilizado ao final da fase de implantação do ciclo de vida do PESC. Após uma iteração completa da evolução do desenvolvimento, o pesquisador terá enfrentado dificuldades e possíveis falhas, bem como terá conseguido sucessos no seu desenvolvimento. Este artefato permite que o desenvolvedor possa registrar as falhas e sucesso obtidos no desenvolvimento de cada versão.

Na primeira seção do artefato, podem ser registradas as falhas de cada iteração. Para tanto, existem as seguintes informações:

- **Descrição da Falha:** A falha ocorrida deve ser descrita nesta seção, com o maior nível de detalhes possível.
- **Prováveis Causas da Falha:** nesta seção, as prováveis causas da falha ocorrida devem ser indicadas.
- **Possível Correção:** devem ser documentadas as medidas adotadas para a correção da falha.

Na segunda seção do artefato, podem ser registradas os sucessos de cada iteração. Para tanto, existem as seguintes informações:

- **Sucesso Alcançado:** O sucesso alcançado deve ser descrito nesta seção, com o nível de detalhes adequado ao desenvolvedor.
- **Pontos relevantes para a obtenção do êxito:** nesta seção, devem ser indicados os passos que permitiram a obtenção do sucesso.

Caso alguma das seções não se aplique à uma determinada versão, ela deverá ser omitida.

4.1.3 Visão Gráfica do Processo Inicial Proposto para o PESC

O ciclo de vida do PESC, com os respectivos artefatos que são produzidos, consultados e/ou alterados, em cada uma de suas etapas, pode ser visto na Figura 4.2.

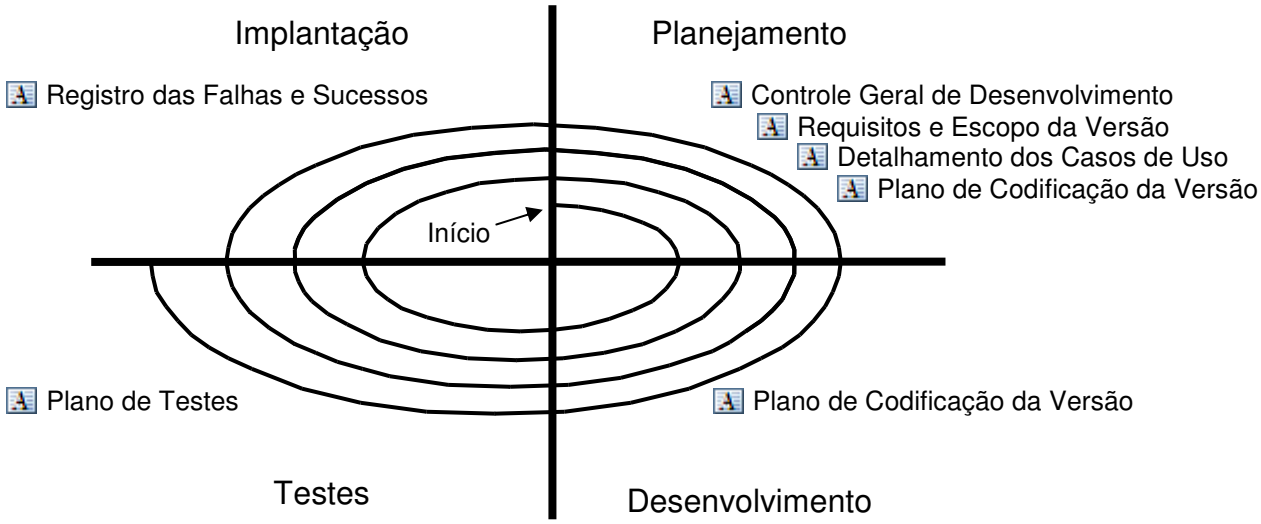

FIGURA 4.2 – Ciclo de Vida do PESC, com seus respectivos artefatos.

Este capítulo apresentou as características do processo inicial do PESC, incluindo o seu ciclo de vida e os artefatos propostos como parte integrante deste processo inicial, com suas informações obrigatórias e opcionais. Foi discutida ainda a forma de como os documentos serão criados e mantidos, bem como a "comunicação" entre eles durante as fases do ciclo de desenvolvimento do PESC.

O próximo capítulo apresenta a análise do processo inicial proposto para o PESC.

5. Análise do Processo Proposto

No capítulo anterior foi caracterizado o processo inicial do PESC, incluindo o seu ciclo de desenvolvimento e artefatos. Este capítulo irá apresentar uma análise do processo inicial do PESC com relação às diretrizes indicadas por Purri (2006).

5.1 Características Indicadas para o Processo

Como demonstrado na seção 3.4 deste trabalho, Purri (2006) gerou várias diretrizes que o processo inicial do PESC deveria contemplar.

A primeira delas diz que o processo inicial do PESC deveria conter um ciclo iterativo e incremental. Esta característica foi atendida, como mostrado na Seção 4.1, que mostra o ciclo iterativo do PESC e suas respectivas fases de cada iteração.

Outra diretriz indicada dizia que o processo inicial deveria ser baseado e/ou adaptado do Processo Unificado (PU). Este processo exerceu uma grande influência na concepção dos artefatos do PESC, a começar do próprio ciclo de desenvolvimento, que também é o ciclo iterativo. Além disso, o PU possui quatro fases em seu ciclo de desenvolvimento. O processo inicial do PESC foi baseado neste ciclo e também apresenta quatro fases, como mostrado e discutido na Seção 4.1.1. Outras características do PU, como a adaptabilidade e as iterações com curto período de tempo, também são notadas no processo inicial do PESC. Sendo assim, o Processo Unificado forneceu a base para a construção do PESC. Vale ressaltar que foram utilizadas ainda várias técnicas idealizadas pela XP, como a simplicidade, a padronização da forma de codificação, a propriedade coletiva sobre o código-fonte, a concepção de pequenas versões e a integração contínua. Sendo assim, os princípios de Extreme Programming também foram de suma importância para a determinação do processo inicial para o PESC.

Outro indicativo era de que o processo inicial deveria ser simples. Nota-se claramente, como mostrado no Capítulo 4 deste trabalho, que o processo é simples o bastante para permitir a sua utilização, sem que o desenvolvedor tenha uma perda considerável de tempo. Ao mesmo tempo, os artefatos do PESC fornecem todas as condições para que o processo de desenvolvimento seja documentado e conseqüentemente apresente um bom padrão de qualidade.

Purri (2006) indicou ainda que o processo deveria ser voltado para uma equipe de desenvolvimento pequena. Este indicativo se relaciona com o de simplicidade e foi naturalmente atendido. O PESC apresenta poucos documentos a serem preenchidos, o que facilita o seu uso por uma equipe de desenvolvimento reduzida.

A UML deveria ser utilizada como base do processo inicial do PESC. Este é outro indicativo gerado por Purri (2006) que foi naturalmente contemplado pois o PESC foi baseado no Processo Unificado, que utiliza esta linguagem como padrão de modelagem.

Uma outra indicação feita por Purri (2006), dizia que o PESC deveria permitir o gerenciamento de código aberto e concorrente. Como já foi dito anteriormente, o PESC idealmente deve ser aplicado através de uma ferramenta automatizada, que deverá permitir este controle de acesso ao código-fonte. No entanto, como o escopo desta parte da pesquisa se limita apenas à definir o processo inicial, e não automatizá-lo, sugere-se a aplicação de uma ferramenta já conhecida e utilizada para gerenciar o acesso ao código-fonte.

Uma outra característica que o processo inicial do PESC deveria atender é a de que seus artefatos deveriam ser opcionais. Como já foi discutido na Seção 4.1.2, o desenvolvedor seleciona os artefatos que mais lhe são adequados e, além disso, pode optar por omitir algumas seções de cada artefato, como visto ao longo do Capítulo 4. Isso faz com que o PESC seja altamente adaptável, atendendo assim a diretriz indicada.

5.2 Características Indicadas para os Artefatos

Assim como foram indicadas algumas características que o processo inicial do PESC deveria contemplar, Purri (2006) também gerou algumas diretrizes que deveriam ser contempladas nos artefatos do PESC, as quais são descritas ao longo desta seção.

A primeira característica dizia que deveria haver uma descrição das funcionalidades gerais do software. Esta característica foi contemplada no artefato "guarda-chuva", *Controle Geral de Desenvolvimento.* Como mostrado na seção 4.1.2.1, este artefato contém uma seção onde o desenvolvedor tem a possibilidade de registrar as funcionalidades gerais do software. O desenvolvedor pode ainda, dentro deste artefato, detalhar uma eventual arquitetura do projeto, característica esta também apontada por Purri (2006) como importante. Uma outra seção deste documento registra o tipo de licença característica do software que será construído, já que esta característica também foi indicada como sendo crucial por Purri (2006). Existe ainda uma outra seção neste documento, denominada de *Descrição da Relevância Científica do Software,* que também foi outra diretriz gerada por Purri (2006).

Uma outra diretriz indicada por Purri (2006) dizia respeito a regras de codificação e documentação do código-fonte. Esta característica é contemplada ainda no artefato *Controle Geral de Desenvolvimento.* Este documento traz uma seção onde o desenvolvedor registra e documenta o padrão de codificação que será adotado durante todo o processo de desenvolvimento do software. A própria documentação do código-fonte se dará de forma natural com a adoção de um padrão de codificação. Sugere-se, como estratégia extra de documentação do código-fonte, que o desenvolvedor comente o seu código, de forma que o seu processamento possa ser entendido apenas pela leitura destes comentários, sem a necessidade de se ler o código-fonte propriamente dito. Dessa forma, garante-se que este código será compreendido por qualquer membro da equipe de desenvolvimento.

Vale ressaltar aqui uma outra característica do artefato *Controle Geral de Desenvolvimento.* Ele possui uma seção que permite que o desenvolvedor documente os módulos que estão previstos para serem desenvolvidos ao longo do projeto, além de possuir uma outra seção onde o desenvolvedor pode efetuar um controle de versões de seus artefatos, gerando assim um histórico da evolução do seu sistema.

Outra característica que o processo inicial deveria atender é a descrição das restrições e viabilidade do projeto. Esta característica foi contemplada no artefato *Requisitos e Escopo da Versão,* que define as seções *Limitações da Versão*, *Restrições de Performance, Restrições de Usabilidade, Adaptabilidade e Portabilidade, Restrições de Hardware e Viabilidade,* como discutido na Seção 4.1.2.2. Estas seções são divididas por cada iteração, onde cada uma das análises de viabilidade e de restrições devem ser feitas para cada versão que será desenvolvida. Vale ressaltar que o desenvolvedor pode optar por omitir as seções que não se aplicarem à versão que será desenvolvida.

O levantamento completo dos requisitos e a sua especificação detalhada são outras características que deveriam ser contemplada nos artefatos do PESC. Estas características podem ser observadas no artefato *Requisitos e Escopo da Versão*, que define a seção de *Descrição dos Requisitos da Versão*. Nesta seção o desenvolvedor tem a possibilidade de registrar todos os requisitos identificados, apresentando o seu detalhamento. Este artefato conta ainda com outras seções, denominadas de *Funcionalidades Esperadas da Versão* e *Escopo da Versão*, que aumentam o nível de detalhes da documentação da versão que será construída.

O diagrama de casos de uso, sua descrição e especificação deveriam estar presentes nos artefatos do processo inicial do PESC, segundo Purri (2006). Para esta característica foi gerado um artefato à parte, denominado de *Detalhamento dos Casos de Uso* que apresenta as seções necessárias para o completo detalhamento deste diagrama e que foram *,* como apresentadas na Seção 4.1.2.3. O diagrama em si deve ser colocado na seção denominada de *Diagrama de Casos de Uso.* O desenvolvedor pode efetuar o seu detalhamento

na seção posterior, denominada de *Detalhamento do Diagrama de Caso de Uso.* Esta seção apresenta os casos de uso, com seus respectivos atores, fluxo de eventos, requisitos especiais, pré-condições, pós-condições e pontos de extensão que podem existir no diagrama.

Outro diagrama que, segundo Purri (2006) também deveria estar contido nos artefatos do PESC é o diagrama de classes. Este característica é contemplada no artefato denominado *Plano de Codificação da Versão,* como pode ser visto na Seção 4.1.2.4. Este artefato apresenta uma seção onde o diagrama de classes é colocado. A seção posterior do documento permite o detalhamento deste diagrama. Para tanto, esta seção apresenta as classes do diagrama, com nome, finalidade, atributos, operações, relacionamentos e responsabilidades de cada uma delas. Outras características previstas por Purri (2006), como a citação das referências científicas nas quais a implementação irá se basear e a descrição dos algoritmos complexos, também são contempladas neste artefato. Estas seções do documento são optativas e devem ser utilizadas somente quando se fizerem necessárias. Este artefato contempla ainda outra característica indicada por Purri (2006), que é a documentação dos componentes reutilizados na construção da versão. O documento apresenta uma seção específica para este fim, onde o desenvolvedor pode detalhar os componentes que foram reutilizados na construção da versão, indicando o seu nome ou identificador, suas responsabilidades e o detalhamento de sua interface. Por fim, este artefato permite ainda que o desenvolvedor detalhe um eventual componente gerado que possa ser utilizado em outros projetos, como indicado por Purri (2006). Para isso, ele conta com uma seção onde é possível o detalhamento deste componente, denominada de *Detalhamento da Interface do Componente.*

O artefato denominado *Plano de Testes* contempla as características de planejamento e descrição dos testes do sistema indicadas por Purri (2006). O desenvolvedor conta com uma série de seções, como mostrado na Seção 4.1.2.5, onde pode ser feito o detalhamento tanto dos testes de código-fonte quanto dos testes de interface, além dos testes de integração, performance e de adequação com os requisitos levantados. Ainda neste artefato, o

desenvolvedor pode registrar os erros encontrados em cada um dos diferentes tipos de testes e suas causas.

Um outro artefato foi criado para registrar as falhas e sucessos do projeto, já que estas características foram indicadas por Purri (2006) como sendo importantes. As informações que compõem este artefato podem ser vistas, na íntegra, na Seção 4.1.2.6. Neste artefato, o desenvolvedor pode registrar as falhas identificadas durante a construção de uma determinada versão, com as suas prováveis causas e possíveis soluções. O desenvolvedor pode ainda optar por registrar um sucesso ou avanço significativo conseguido no desenvolvimento da versão. Neste caso, o desenvolvedor deve registrar este sucesso obtido, além de indicar os pontos relevantes para que este pudesse ser obtido. Este registro é importante, pois permite a criação de um banco de conhecimento sobre os êxitos e, principalmente, sobre as falhas ocorridas no desenvolvimento, para que outros membros da equipe que eventualmente necessitem continuar o projeto não cometam os mesmos erros já encontrados por algum pesquisador.

Uma última característica importante citada por Purri (2006) é a confecção de um manual de uso do sistema que for gerado. Este ponto não foi incluído nos artefatos do PESC, ficando a cargo do próprio desenvolvedor a definição do seu *layout* e do seu conteúdo. Recomenda-se que este manual seja o mais completo possível e que apresente uma descrição de todas as funcionalidades do software de forma detalhada, para facilitar o seu uso, bem como suas limitações e restrições, identificadas ao longo do ciclo de desenvolvimento do software.

Este capítulo apresentou um cruzamento das diretrizes indicadas por Purri (2006) e das características correspondentes presentes no PESC e nos seus artefatos. O próximo capítulo apresenta as considerações finais deste trabalho.

6. Conclusão

6.1 Conclusões e Considerações Finais

Esta pesquisa se propôs a contribuir com o processo de desenvolvimento de software científico de natureza acadêmica, com a intenção de conceber um processo inicial para o PESC, com o intuito de conseguir para o software científico padrão similar de qualidade ao esperado do software convencional. O trabalho procurou caracterizar os processos de desenvolvimento específicos para o desenvolvimento de software comerciais, bem como mostrou a evolução da pesquisa, que levou à concepção do processo inicial para o PESC – *Processo de desenvolvimento Específico para Software Científico*.

O objetivo principal do trabalho, que era conceber um processo inicial para o PESC, foi atingido.

Foi feita uma detalhada revisão bibliográfica sobre os processos de desenvolvimento de software já existentes, bem como das diretrizes indicadas por Purri (2006) para a concepção do processo inicial.

Posteriormente, o processo inicial foi concebido. Com base nas diretrizes geradas, e com o apoio da literatura, foram criados os artefatos do processo inicial do PESC. Purri (2006), em uma das suas diretrizes, indicou que o PU (Processo Unificado) deveria ser tomado como base para o processo inicial. No entanto, com a observância da literatura, foi verificado que a XP (Extreme Programming) é uma técnica bastante promissora, e possui características interessantes que poderiam ser utilizadas no PESC. Partindo deste princípio, o processo inicial foi fortemente fundamentado nestes dois processos de desenvolvimento de software convencionais.

Vale salientar que os artefatos do processo inicial do PESC compreendem as diretrizes previamente conhecidas. Cada um dos artefatos compreende uma parte do desenvolvimento do software. Estes artefatos foram criados de forma

que pudessem apresentar um alto grau de simplicidade, para facilitar o entendimento e a utilização do processo. Não existem pontos formais de validação ou aceitação dentro do processo, fato este que agiliza o desenvolvimento. Alguns artefatos são ainda opcionais, assim como algumas de suas seções. Eles compõem um arcabouço, onde o desenvolvedor seleciona e utiliza apenas os artefatos que melhor atendem a sua real situação. Dentro de cada artefato selecionado, o desenvolvedor tem a possibilidade ainda de definir quais as seções do documento são mais pertinentes ao seu desenvolvimento, o que torna o PESC um processo altamente adaptativo e customizável, além de extremamente simples. Esta facilidade de uso do processo permite que o software construído coopere com diversos projetos de pesquisa, de forma confiável e ordenada. Além disso, permite que estes software científicos possam ser adequadamente estudados, utilizados, modificados e redistribuídos.

O enfoque deste trabalho foi a concepção de um processo inicial para o PESC. Sendo assim, esta pesquisa concentrou-se em definir este processo de acordo com as diretrizes indicadas e com o apoio da literatura. Não faz parte deste trabalho a construção de uma ferramenta automatizada que implemente o processo inicialmente definido, tampouco a validação do processo, sendo estas tarefas citadas na próxima seção como trabalhos futuros.

6.2 Trabalhos Futuros

Visando a continuação da pesquisa, algumas idéias são apontadas a seguir:

O próximo passo deste projeto de pesquisa é efetuar o passo sete da metodologia de Humphrey, que é a validação formal do processo inicial do PESC. Humphrey (1995) sugere que o ideal é tentar testar o processo em projetos pequenos ou pilotos para, então, refinar e modificar o processo de acordo com os resultados dos testes realizados. Esta validação deve focar a área de ciências exatas e de engenharia, já que são as áreas focadas pelo processo inicial do PESC. De posse das informações de melhoria coletadas

durante esta validação, o processo deve ser melhorado, incorporando estas melhorias, gerando assim um processo mais refinado, que atenda de forma mais realística às necessidades da comunidade científica que desenvolve software. A melhoria do processo contempla a última etapa da metodologia de Humphrey (1995). A validação formal do PESC contribuirá para que ele possa efetivamente ser aceito como um processo que auxilia o pesquisador na documentação de seus software e conseqüentemente no seu desenvolvimento cotidiano.

Uma etapa relevante para a pesquisa seria a de efetuar uma validação do PESC em outras áreas do conhecimento, para verificar a sua aplicabilidade. As observações coletadas em outras áreas do conhecimento podem também ser importantes para o desenvolvimento do PESC.

Após a validação e melhoria do processo, a construção de uma ferramenta automatizada que implemente e automatize o processo é um importante passo rumo à difusão do PESC. Esta ferramenta deve preferencialmente ser acessível via web, para que todos os membros da equipe tenham acesso facilitado aos dados do projeto. A ferramenta deverá efetuar o controle sobre os documentos gerados durante o desenvolvimento de um determinado sistema. Vale salientar que, com a automatização do PESC, o artefato guarda-chuva praticamente deixará de existir, ficando a cargo da própria ferramenta os controles por ele gerenciados. Após a criação da ferramenta web, devem ser feitos testes tradicionais e posteriormente deve-se divulgar esta importante contribuição à comunidade científica.

REFERÊNCIAS

AMBLER, S. W. *An Introduction to Process Patterns*. 1998. Disponível em: http://www.ambysoft.com/downloads/processPatterns.pdf - Visitado em: 17/08/2007.

ASTELS D., MILLER G., NOVAK M., eXtreme Programming: Guia prático, Campus, Rio de Janeiro, 2002.

BECK K., *Extreme Programming Explained*. Boston, MA: Addison-Wesley, 2000.

BELLOQUIM, A., *PSP – O Processo de Software Pessoal*. 1999. Disponível em: http://www.choose.com.br/artigos/html/textos/dm1098.htm - Visitado em 05/05/2007.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *UML Guia do Usuário*, Elsevier, Rio de Janeiro, 2000.

CORDEIRO, M. A., *Foco no processo*. 2000. Companhia de Informática no Paraná – CELEPAR. Disponível em: http://www.celepar.br/batebyte/bb100/foco.htm - Visitado em 28/04/2007.

FOWLER M., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman, 1999

GUEDES G.T.A , *UML: uma abordagem prática.* 2 ed. São Paulo: Novatec, 2006.

HUMPHREY, W. S. *The Personal Software Process. Software Process Newsletter*, Technical Council on Software Engineering, IEEE Computer Society, Volume 13, N. 1. Setembro 1994. pp SPN 1-3.

HUMPHREY, W. S, A *Discipline for Software Engineering*. Addison-Wesley Reading-MA, 1995.

IEEE Standards Collection: *Software Engineering*, IEEE Standard 610.12 – 1990, IEEE, 1993.

JACOBSON, I.; RUMBAUGH, J.; BOOCH, G. *The Unified Software Development Process*. Addison-Wesley, Reading – MA, 1999.

JEFFRIES R. *What is Extreme Programming?* Disponível em: http://www.xprogramming.com/xpmag/whatisxp.htm - Visitado em: 15/08/2007.

KRUTCHEN, P., *The Rational Unified Process: An Introduction*. 3rd ed. Addison-Wesley. 2003

KRUCHTEN, P., *What is the Rational Unified Process?*. Copyright Rational Software 2001. Disponível em: http://www-106.ibm.com/developerworks/rational/library/ content/RationalEdge/jan01/WhatIstheRationalUnifiedProcessJan01.pdf - Visitado em 29/03/2007.

LARMAN, C. *Applying UML and Patterns: an introduction to Object-oriented Analysis And Design and The Unified Process*. 2 ed. Prentice Hall, 2002.

LARMAN, C. *Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos e ao Processo Unificado.* Trad. Luiz Augusto Meirelles Salgado e João Tortello. 2 ed. Porto Alegre: Bookman, 2004.

PAULA, W. P. *Engenharia de Software – Fundamentos, Métodos e Padrões.* LTC Editora. Rio de Janeiro - RJ, 2001.

PAULA, W. P. *Processo Práxis*. Disponível em: http://www.wppf.uaivip.com.br/praxis - Visitado em: 28/07/2007

PAULA, W. P. *A Process-based Software Engineering Course: Some Experimental Results.* In: Proceedings of the 3as. Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento. Valdivia, Chile, Nov. 2003

PEREIRA Jr, M.; PURRI, M. C. M. S.; MOITA, G. F. *Definição e Validação de um Processo de Desenvolvimento para Software Científico.* XXVIII CILAMCE – Iberian Latin American Congress on Computational Methods in Engineering. Junho 2007. Porto-Portugal.

PRESSMAN, R. S. *Engenharia de Software*. 5º ed. Rio de Janeiro: McGraw-Hill, 2002.

PRESSMAN, R. S. *Engenharia de Software*. 6º ed. Rio de Janeiro: McGraw-Hill, 2006.

PURRI, M. C. M. S. *Estudo e Propostas Iniciais para a Definição de um Processo de Desenvolvimento para Software Científico.* Dissertação de Mestrado. Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG. Belo Horizonte, 2006.

PURRI, M. C. M. S; PEREIRA Jr, M.,; MOITA, G. F. *PESC – Processo de Desenvolvimento Específico para Software Científico: Propostas Iniciais.* XXVII CILAMCE – Iberian Latin American Congress on Computational Methods in Engineering. Setembro 2006. Belém-PA.

REZENDE, DENIS. A., *Engenharia de Software e Sistemas de Informação*. 3ª ed. Rio de Janeiro: Brasport, 2005.

RUP – *Rational Unified Process Guide*. Disponível em: http://www. wthreex.com/rup - Visitado em: 25/02/2007.

SHALLOWAY A., TROTT J. R. *Explicando Padrões de Projeto: uma nova perspectiva em projeto orientado a objeto.* Trad. Ana M. de Alencar Price. Porto Alegre: Bookman, 2004.

SILVA, A. A., GOMIDE C. F., PETRILLO F., *Metodologia e projeto de software orientados a objetos: modelando, projetando e desenvolvendo sistemas com UML e componentes distribuídos.* 1ª ed. São Paulo: Érica, 2003.

SOMMERVILLE I., *Engenharia de Software.* Trad. André Maurício de Andrade. São Paulo: Addison Wesley, 2003.

WASLAWICK, R. SIDNEI. *Análise e projeto de sistemas de informação orientados a objetos.* Rio de Janeiro: Elsevier, 2004.

WIKIPEDIA A enciclopédia livre. Disponível em: http://www.wikipedia.com.br.

Anexo A

Instituição de Pesquisa
Nome do Projeto
Versão:

Controle Geral de Desenvolvimento

Este documento contém o registro das informações gerais do software e das alterações efetuadas ao longo do seu desenvolvimento

Mês/Ano

Informações Gerais

Nome do Software:

Nesta parte deve ser colocado o nome do software que está sendo desenvolvido.

Integrantes/Desenvolvedores:

Nesta parte deve ser colocado o nome do(s) desenvolvedor(es) do projeto.

Definição da Licença do Software:

Nesta parte deve ser indicada o tipo de licença característica do software desenvolvido.

Funcionalidades Gerais do Software:

Nesta parte deve ser feita uma descrição das funcionalidades gerais do software a ser desenvolvido.

Descrição da Relevância Científica do Software:

Nesta parte deve ser feita a descrição da relevância científica do software que será desenvolvido.

Padrão de Codificação:

Esta parte do documento deve ser preenchida com o padrão de codificação que será adotado ao longo do ciclo de desenvolvimento do software. Aqui o desenvolvedor define as formas que os comandos presentes no código-fonte terão. Por exemplo:

- *Quebra de linha*

Procure sempre usar linhas com no máximo 80 caracteres.

Estruturas de controle (if, for, while, switch, etc)
Sempre coloque um espaço entre a estrutura de controle e o parenteses.
Use parenteses mesmo nas situações em que eles forem opcionais.

- *Convenção de Nomes*

Para classes

As classes devem ter nomes descritivos, procurando evitar abreviações onde for possível.
Todo nome de classe deve começar com uma letra maiuscula.
Se existir hierarquia entre as classes, ela deve ser especificada separando cada nível da hierarquia com um underscore "_".

Módulos

Módulos:

Nesta parte devem ser registrados os dados de cada módulo que o software a ser desenvolvido conterá.

- *Nome do Módulo*

  Esta parte deve conter o nome do módulo.
- *Responsável(is)*

  Esta parte deve conter o nome do(s) responsável(is) pelo desenvolvimento do módulo.
- *Descrição Geral do Módulo*

  Esta parte deve conter uma descrição geral do módulo a ser desenvolvido.
- *Início do Desenvolvimento*

  Esta parte deve registrar a data do início do desenvolvimento do software.

Detalhamento da Arquitetura do Projeto
Arquitetura do Projeto: *Nesta parte deve ser colocada a figura da arquitetura do projeto.*
Descrição: *Nesta parte deve ser feita a descrição da arquitetura apresentada acima.*

Controle de Versões
Nome do Artefato: *Nesta parte deve ser colocado o nome do artefato que sofreu a alteração que será registrada.*
Versão do Artefato que sofreu a alteração: *Nesta parte deve ser registrada a versão do artefato que foi alterada. Deve-se indicar ainda se esta modificação gerou uma nova versão. Caso tenha gerado, esta nova versão deve também ser informada nesta sub-seção.*
Data da Alteração: *Nesta parte deve ser colocada a data da alteração feita no artefato.*
Autor da Alteração: *Nesta parte deve ser colocado o nome do autor que efetuou a alteração.*
Alterações efetuadas: *Esta parte apresenta e detalha as alterações efetuadas em cada artefato.*

Instituição de Pesquisa
Nome do Projeto
Iteração:

Requisitos e Escopo da Versão

Este documento contém o registro dos requisitos identificados para o desenvolvimento da versão

Mês/Ano

Requisitos e Escopo da Versão

Descrição do Requisitos da Versão:

Nesta seção devem ser colocados os principais pontos deste documento.

A identificação bem feita dos requisitos de um sistema é parte crucial em um desenvolvimento de sucesso. Existem várias técnicas para a elucidação dos requisites do software a ser desenvolvido. Uma das mais simples é a aplicação de questionários às pessoas envolvidas. No entanto, a utilização de questionários é uma tarefa bastante formal e, exatamente por esta característica, não se aplica completamente ao PESC, já que este processo é voltado para o desenvolvimento de software científicos por pequenas equipes. O que geralmente acontece em um desenvolvimento deste porte, são pequenas discussões entre os membros do grupo de pesquisa para a definição dos requisitos do sistema a ser desenvolvido. Estas particularidades identificadas nestas discussões devem ser registradas neste documento.

Após identificados os primeiros requisitos do software a ser desenvolvido, a construção de protótipos é uma das técnicas de engenharia de requisitos que deve ser empregada. Esta é uma técnica bastante valiosa que permite o contato do usuário que, no enfoque do PESC, é o próprio pesquisador e/ou orientador, com uma parte executável do sistema, coletando assim suas observações, sob um ponto de vista mais prático. Com a visualização e utilização de um protótipo, o pesquisador ou orientador tem condições de apontar os passos da próxima etapa do desenvolvimento.

Cabe aqui ressaltar que um protótipo tem a única função de permitir um contato real do usuário com o sistema ou parte dele, permitindo a identificação de novos requisitos, não levando em conta os padrões de qualidade exigidos em um sistema real. Os protótipos podem ser, inclusive, desenvolvidos em ferramentas de quarta geração, para agilizar a identificação de requisitos. Depois de identificadas as necessidades do sistema, o protótipo deve ser descartado e, com base neste, um novo programa deve ser iniciado, agora observando os padrões de qualidade para garantir um software de qualidade.
Depois da coleta de requisitos através de reuniões e/ou discussões e protótipos, uma descrição detalhada do software a ser construído deve ser feita nas respectivas sub-seções abaixo

Funcionalidades Esperadas da Versão:

Esta sub-seção apresenta as principais funcionalidades da versão, com o nível de detalhamento adequado ao desenvolvedor.

Escopo da Versão:

Nesta sub-seção devem ser mostrados os pontos que serão abrangidos pela versão que está sendo desenvolvida nesta iteração.

Início do Desenvolvimento:

Nesta sub-seção o desenvolvedor deve registrar a data em que a construção do módulo foi iníciada..

Limitações da Versão:

Esta sub-seção apresenta as limitações da versão, com o nível de detalhamento adequado ao desenvolvedor e orientador.

Restrições de Performance:

Caso existam restrições acerca da performance que a versão deve atingir, estas devem ser descritas nesta sub-seção. Caso não existam tais restrições, esta sub-seção deve ser omitida.

Um exemplo de restrição de performance, seria a necessidade de a versão efetuar um determinado cálculo em tempo hábil, ou seja, não basta que a versão funcione, mas ele não deve demorar mais que 10 minutos para efetuar um determinado processamento, pois isso poderia comprometer uma parte posterior do processo.

Restrições de Usabilidade, Adaptabilidade e Portabilidade:

Caso existam restrições de usabilidade, adaptabilidade e portabilidade que a versão deve anteder, estas devem ser descritas nesta sub-seção. Caso não existam tais restrições, esta sub-seção deve ser omitida.

Por exemplo, se o sistema for desenvolvido para comandar um robô, ele terá uma restrição de

adaptabilidade, pois deverá ser compatível com o hardware existente.

Se o sistema tiver a necessidade de ser multi-plataforma, ele terá uma restrição de portabilidade, já que deverá funcionar em qualquer ambiente.
Como exemplo de restrição de usabilidade, podemos citar o emprego de vários atalhos distintos dentro do sistema para a execução de uma mesma tarefa.

Restrições de Hardware:

Caso existam restrições acerca do hardware a ser utilizado pela versão, estas devem ser descritas nesta sub-seção. Caso não existam tais restrições, esta sub-seção deve ser omitida.

Como exemplo de restrições de hardware, podem ser citados os algoritmos heurísticos, que podem exigir alto poder de processamento e de memória, dependendo dos dados a serem processados. Nestes casos,onde existe uma alta necessidade de recursos, freqüentemente utiliza-se um cluster de computadores.

Restrições como esta devem ser descritas nesta seção.

Viabilidade:

Caso seja necessário indicar a viabilidade do projeto em questão, esta deve ser descrita nesta sub-seção. Caso não exista a necessidade da análise de viabilidade do projeto, esta sub-seção deve ser omitida.

Instituição de Pesquisa
Nome do Projeto
Iteração:

Detalhamento dos Casos de Uso

Este documento apresenta o Diagrama de Casos de Uso, bem como o seu detalhamento.

Mês/Ano

Detalhamento dos Casos de Uso

Diagrama de Casos de Uso:

Um diagrama de Caso de Uso é de suma importância em um desenvolvimento de software moderno, pois permite que os requisitos identificados sejam confrontados com as funcionalidades representadas em cada caso de uso. Sendo assim, o sistema pode ser testado antes mesmo de ser implementado.

Um sistema pode possuir mais de um diagrama de Caso de Uso. Sendo assim, cada um deles deve ser explicitado replicando-se todas as seções deste documento.

Em cada iteração do ciclo de vida, pode-se gerar um diagrama de Casos de Uso, caso seja necessário.

Esta sub-seção apresenta o diagrama de caso de uso principal.

Detalhamento do Diagrama de Casos de Uso:

Nesta sub-seção devem ser detalhados todos os casos de uso mostrados no diagrama principal, segundo o esquema abaixo:

- *Nome do Caso de Uso*

  Esta sub-seção deve conter o nome do caso de uso e sua finalidade dentro do diagrama.

  - *Ator(es)*

    Nesta sub-seção devem ser explicitados o(s) ator(es) ligados ao caso de uso em questão.

  - *Fluxo de Eventos*

    Esta sub-seção apresenta uma descrição textual que mostra como cada caso de uso é realizado, em uma seqüência de eventos. Engloba os eventos que estão explicitados no diagrama de Caso de Uso Principal. Esta seção oportunamente pode ser dividida em Fluxo Básico e Fluxos Alternativos. O Fluxo Básico descreverá a ação feita pelo ator correspondente e a conseqüente resposta do sistema. Os Fluxos Alternativos apresentam alternativas ao Fluxo Básico, podendo representar, por exemplo, exceções que acontecem no Fluxo Básico.

  - *Requisitos Especiais*

    Esta sub-seção apresenta os requisitos especiais do caso de uso em questão. Um requisito especial é específico a um determinado caso de uso, mas não pode ser representado no diagrama. Exemplos de requisitos especiais incluem requisitos legais, atributos de qualidade (usabilidade, performance, etc), restrições de plataforma de hardware e sistema operacional.

  - *Pré-condição*

    Esta sub-seção apresenta a(s) pré-condição(ões) do caso de uso em questão. Uma pré-condição de um caso de uso é o estado que o sistema deve apresentar antes deste caso de uso ser executado.

  - *Pós-condição*

    Esta sub-seção apresenta a(s) pós-condição(ões) do caso de uso em questão. Uma pós-condição de um caso de uso é o estado, ou uma lista de estados, que o sistema irá apresentar imediatamente após a finalização do caso de uso.

  - *Pontos de Extensão*

    Esta sub-seção apresenta o(s) ponto(s) de extensão do caso de uso em questão.

Instituição de Pesquisa
Nome do Projeto
Iteração:

Plano de Codificação da Versão

Este documento contém o registro do plano de codificação da versão

Mês/Ano

Plano de Codificação da Versão

Nome ou identificador da versão:

Nesta sub-seção deve ser descrito o nome ou identificador da versão que será produzida nesta iteração.

Diagrama de Classes:

Nesta sub-seção deve ser inserido o Diagrama de Classes do Módulo.

Detalhamento do Diagrama de Classes:

Os tópicos abaixo devem ser replicados para cada classe existente no Diagrama de Classes mostrado acima.

- *Nome e finalidade da Classe*

  Esta sub-seção deve conter o nome da classe e sua finalidade dentro do diagrama.

  - *Atributos*

    Nesta sub-seção devem ser relacionados os atributos de cada classe, incluindo nome, tipo e importância perante os requisitos definidos.

  - *Operações*

    As operações da classe devem ser descritas aqui, incluindo nome, breve descrição, argumentos e detalhes de seu funcionamento.

  - *Relacionamentos*

    Nesta sub-seção devem ser apresentados os relacionamentos que esta classe possui, explicitando o seu funcionamento de forma clara e concisa.

  - *Responsabilidades*

    Esta sub-seção deve elucidar as responsabilidades da classe.

Componentes Reutilizados por esta Versão:

Esta seção deve conter o detalhamento dos componentes que são reutilizados na construção desta versão.
Os tópicos abaixo devem ser replicados para cada componente que for reutilizado na construção da versão.

- *Nome ou identificador do Componente*

  Esta sub-seção deve conter o nome do componente.

  - *Responsabilidades*

    Esta sub-seção deve elucidar as responsabilidades do componente que está sendo reutilizado.

  - *Interface de Comunicação*

    Nesta sub-seção, a interface de comunicação entre o componente e a versão que está sendo desenvolvida deve ser detalhada, incluindo os métodos e parâmetros disponíveis.

Descrição de Algoritmos Complexos:

Esta seção é opcional e deve ser utilizada quando se fizer necessária a descrição de um algoritmo que seja complexo.

Referências Científicas:

Esta seção é opcional e deve ser utilizada sempre que se fizer necessário registrar uma determinada referência científica em que uma determinada implementação irá se basear.

Detalhamento da Interface do Componente:

Caso seja gerado um componente que possa ser reutilizado, ele deve ser detalhado nesta seção, indicando os métodos e parâmetros que permitem à comunicação de dados entre a versão e o ambiente onde ela será inserida.

Instituição de Pesquisa
Nome do Projeto
Iteração:

Plano de Testes

Este documento contém o registro dos testes efetuados no sistema

Mês/Ano

Plano de Testes
Versão a ser testada: *Nesta seção deve ser registrado o nome do módulo onde o teste será aplicado.*
Testes de Código-Fonte
Objetivo do Teste: *Nesta sub-seção devem ser mostrados os objetivos do teste que será aplicado.*
Técnica de Teste: *Esta sub-seção deve explicitar a técnica de teste que será aplicada.* *Os testes relacionados ao código-fonte do software são chamados de Testes de Caixa-Branca. Existem vários métodos de Caixa-Branca. A técnica de teste que será utilizada deve ser detalhada nesta seção.*
Critérios de Sucesso do Teste: *Esta sub-seção deve conter os critérios que devem ser alcançados para que o teste seja considerado um sucesso.*
Resultados Obtidos: *Esta sub-seção deve conter os resultados obtidos após a execução do teste. Estes resultados poderão ser comparados com os critérios de sucesso, explicitados acima, para verificar se o teste feito obteve sucesso. Caso seja constatado algum erro, este deve ser documentado nesta seção e posteriormente corrigido.*
Testes de Interface
Objetivo do Teste: *Nesta sub-seção devem ser mostrados os objetivos do teste que será aplicado.*
Técnica de Teste: *Esta sub-seção deve explicitar a técnica de teste que será aplicada.* *Os testes relacionados à interface do software são chamados de Testes de Caixa-Preta. Existem vários métodos de Caixa-Preta. A técnica de teste que será utilizada deve ser detalhada nesta seção.*
Dados de Entrada: *Esta sub-seção deve conter os dados de entrada que serão utilizados no teste.*
Dados esperados de Saída (Critérios de Sucesso do Teste): *Esta sub-seção deve conter os dados de saída que são esperados, com base nos dados de entrada acima informados.*
Resultados Obtidos: *Esta sub-seção deve conter os dados de saída que foram gerados após a realização do teste. Estes resultados poderão ser comparados com os dados de saída esperados, explicitados acima, para verificar se o teste feito obteve sucesso. Caso seja constatado algum erro, este deve ser documentado nesta seção e posteriormente corrigido.*
Testes de Integração
Objetivo do Teste: *Nesta sub-seção devem ser mostrados os objetivos do teste que será aplicado.* *Fatores como a compatibilidade com versões anteriores e a garantia de que o restante do programa não será afetado deverão fazer parte desta seção.*
Técnica de Teste: *Esta sub-seção deve explicitar a técnica de teste que será aplicada.*
Critérios de Sucesso do Teste: *Esta sub-seção deve conter os critérios que devem ser alcançados para que o teste seja considerado um sucesso.*

Resultados Obtidos:

Esta sub-seção deve conter os resultados obtidos após a execução do teste. Estes resultados poderão ser comparados com os critérios de sucesso, explicitados acima, para verificar se o teste feito obteve sucesso. Caso seja constatado algum erro, este deve ser documentado nesta seção e posteriormente corrigido.

Testes de Performance

Objetivo do Teste:

Nesta sub-seção devem ser mostrados os objetivos do teste que será aplicado.

Fatores como a compatibilidade com versões anteriores e a garantia de que o restante do programa não será afetado deverão fazer parte desta seção.

Técnica de Teste:

Esta sub-seção deve explicitar a técnica de teste que será aplicada.

Critérios de Sucesso do Teste:

Esta sub-seção deve conter os critérios que devem ser alcançados para que o teste seja considerado um sucesso.

Resultados Obtidos:

Esta sub-seção deve conter os resultados obtidos após a execução do teste. Estes resultados poderão ser comparados com os critérios de sucesso, explicitados acima, para verificar se o teste feito obteve sucesso. Caso seja constatado algum erro, este deve ser documentado nesta seção e posteriormente corrigido.

Testes de Validação de Requisitos

Requisito X Funcionalidade:

Nesta sub-seção, o desenvolvedor deve verificar se as funcionalidades deste módulo estão de acordo com os requisitos elucidados durante a análise de requisitos.

Instituição de Pesquisa
Nome do Projeto
Iteração:

Sucessos e Falhas da Iteração

Este documento contém o registro de todos os sucessos e falhas do projeto

Mês/Ano

Sucessos e Falhas da Iteração
Nome ou identificador da Versão: *Nesta sub-seção deve ser indicado a versão em que houve a falha. Caso haja mais de uma falha em uma mesma versão, as sub-seções abaixo devem ser replicadas para registrar cada uma delas.*
Falhas da iteração
Descrição da Falha: *Nesta sub-seção deve ser descrita a falha ocorrida, com detalhes.*
Prováveis Causas da Falha: *Nesta sub-seção devem ser elucidados todos os pontos que causaram a falha identificada.*
Possível Correção: *Nesta sub-seção o pesquisador deve elucidar como a falha informada acima foi ou pode ser corrigida.*
Sucessos da Iteração
Sucesso Alcançado: *Nesta sub-seção, o sucesso conseguido deve ser descrito, com detalhes.*
Pontos relevantes para a obtenção do êxito: *Nesta sub-seção devem ser elucidados todos os pontos que possibilitaram o sucesso de uma versão.*