UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CAROLINE CARBONELL CINTRA

A Implantação de um Processo de Engenharia de Requisitos Baseado no Processo Unificado da Rational (RUP) Alcançando Nível 3 de Maturidade da Integração de Modelos de Capacidade e Maturidade (CMMI) Incluindo a Utilização de Práticas de Métodos Ágeis

Dissertação apresentada como
requisito parcial para obtenção do grau
de Mestre em Ciência de Computação

Prof. Dr. Roberto Tom Price
Orientador

Porto Alegre, abril de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cintra, Caroline Carbonell

A Implantação de um Processo de Engenharia de Requisitos Baseado no Processo Unificado da Rational (RUP) Alcançando Nível 3 de Maturidade da Integração de Modelos de Capacidade e Maturidade (CMMI) Incluindo a Utilização de Práticas de Métodos Ágeis / Caroline Carbonell Cintra. – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

160f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2006. Orientador: Roberto Tom Price.

1. Processo de desenvolvimento de Software. 2. Engenharia de Requisitos. 3. Gerência de Requisitos. 4. CMMI. 5. RUP. 6. Métodos Ágeis. I. Price, Roberto Tom. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profa. Valquiria Linck Bassani
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Flávio Rech Wagner
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Ao Professor Roberto Tom Price, meu orientador, agradeço pela disponibilidade constante para debater, sugerir, revisar, instigar, criticar, enfim, conduzir este trabalho compartilhando comigo suas visões sobre engenharia de *software* e produção acadêmica.

Aos colegas e amigos Júlio Hartmann, Lisandra Manzonni Fontoura, Viviane Rangel, José Henrique Amaral dos Santos, Vanessa Pires e Flávio Knob, e todo o *Grupo de Processo de Software* da DBServer, que contribuíram com este trabalho deixando suas próprias pesquisas à minha disposição, agradeço com a reverência de quem reconhece o trabalho científico criterioso e de qualidade.

À DBServer Assessoria em Sistemas de Informação, que proporcionou a experimentação do processo de engenharia de requisitos aqui definido, agradeço pelo espaço aberto a este trabalho e pelas inúmeras experiências profissionais que, de um modo ou de outro, estão refletidas ao longo deste volume. Em especial, agradeço aos sócios gerentes Eduardo Peres, Verner Heidrich e Mário Bastos, por tornarem este trabalho possível. E ao Eduardo, agradeço por todo o conhecimento compartilhado, pelas inúmeras lições aprendidas em engenharia de requisitos e por todas as contribuições à minha visão atual deste assunto – e de tantos outros!

A todas as pessoas a quem eu faltei para que pudesse concretizar este trabalho, agradeço pela compreensão e por não terem desistido de mim mesmo depois de tanta ausência. À minha família, especialmente à minha mãe Lisiene Maria Carbonell Cintra e aos meus avós Léa e Maurity Carbonell, sem esquecer dos Rech e Santos, agradeço por não questionarem as minhas inúmeras faltas e, pior do que isto, o meu constante estado de desespero nestes dois anos.

Por fim, ao meu amigo, colega, consultor, revisor, investidor, responsável pelo apoio logístico e psicológico, ou seja, ao maior *stakeholder* deste trabalho, Rafael Rech, agradeço por tudo; por ser, para mim, o porto seguro em todas as tempestades – e esta foi das grandes!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CMM	*Capability Maturity Model.*
CMMI	*Capability Maturity Model Integration*
ERP	*Enterprise Resource Planning*
FURPS	*Functionality, Usability, Reliability, Performance, Supportability*
HTML	*Hypertext Markup Language*
PA	*Process Area* (Área de Processo)
PMT	*Pattern-based Methodology Tailoring*
RFP	*Request for Proposals*
RUP	*Rational Unified Process* (Processo Unificado Rational)
SEI	*Software Engineering Institute.*
SW-CMM	*Software Capability Maturity Model*
UML	*Unified Modeling Language.*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho descreve a definição e institucionalização de um processo de engenharia de requisitos que está em conformidade com as áreas de processo do CMMI (*Capability Maturity Model*) de Gerência de Requisitos e Desenvolvimento de Requisitos e cujos componentes (atividades, papéis, produtos de trabalho) são baseados em RUP (*Rational Unified Process*).

A principal contribuição deste estudo é a definição de um processo de engenharia de requisitos baseado em abordagens de desenvolvimento diferenciadas, que foi implantado em uma organização específica, com foco em praticidade, eficiência e retorno do investimento. A implantação do processo em projetos reais permitiu sua experimentação, avaliação e refinamento, validando as alternativas de integração utilizadas para empregar as abordagens de desenvolvimento escolhidas.

Complementando o processo proposto, como decorrência do foco em eficiência, são consideradas possibilidades de emprego de práticas de métodos ágeis na execução do processo, com o intuito de aumentar a produtividade do mesmo, sustentando sua garantia de qualidade.

O processo proposto é descrito, do método de concepção aos passos envolvidos e artefatos gerados em cada atividade. Também são comentadas as etapas e áreas de trabalho envolvidas na institucionalização do trabalho.

Palavras-Chave: CMMI, *Capability Maturity Model Integration*, RUP, *Rational Unified Process*, métodos ágeis, processo de desenvolvimento de *software*, engenharia de requisitos, gerência de requisitos, desenvolvimento de requisitos, institucionalização.

Implanting a Requirements Engineering Process based on Rational Unified Process (RUP) reaching Capability Maturity Model Integration (CMMI) Maturity Level 3 and including Agile Methods Practices

ABSTRACT

This research depicts the definition and institutionalization of a requirements engineering process which is in conformance to CMMI (*Capability Maturity Model*) Requirements Management and Requirements Development process areas. The proposed process components (activities, roles, work products) are based on *Rational Unified Process* (RUP) process framework.

The proposed process main contribution is the definition of a requirements engineering process, leveraging such diverse development approaches, which was implemented in a specific organization, focusing on practicality, efficiency and return on investment. Implementing such process in real projects has promoted its experimentation, evaluation and refinement, validating the integration alternatives used to bring together the chosen development approaches.

The possibility of employing agile methods practices through the process execution is discussed, aiming at increasing the process productivity, while assuring product quality.

The proposed process details are described, from method conception to each activity steps and generated artifacts. The process institutionalization phases and work areas are also commented.

Keywords: CMMI, Capability Maturity Model Integration, RUP, Rational Unified Process, agile methods, software development process, requirements engineering, requirements management, requirements development, institutionalization.

1 INTRODUÇÃO E MOTIVAÇÃO

1.1 Motivação e Definição do Trabalho

1.1.1 Panorama Atual dos Processos de Desenvolvimento de Software

A percepção da crise de *software,* no final dos anos 60, resultou na criação de processos bem definidos para o desenvolvimento de *software*, baseados na identificação de fases e estágios. Estes processos, cuja idéia central foi o *Modelo em Cascata* ou *Waterfall Model* (ROYCE, 1970), foram largamente adotados e contribuíram para o sucesso de diversos projetos.

Com o passar do tempo e com a evolução dos ambientes tecnológico e de negócios, novas metodologias de desenvolvimento de *software* foram criadas, incluindo-se aí diversas variações de processos incrementais e iterativos, ou seja, processos nos quais o *software* é desenvolvido em iterações, cada qual produzindo uma versão do sistema que complementa a anterior (LARMAN; BASILI, 2003).

Embora a preocupação com a engenharia de *software* seja evidente na literatura, com inúmeros estudos relacionados a processos de desenvolvimento, o panorama dos projetos de *software* continua enfrentando sérios problemas na busca pelo desenvolvimento bem sucedido ainda nos dias de hoje. Em (LYCETT et al., 2003), menciona-se um relatório de 1996 sobre o desempenho do mercado de Tecnologia da Informação (CLEGG et al., 1996), de acordo com o qual "*de 80 a 90% do software não atinge seus objetivos de performance, 80% dos sistemas são entregues atrasados e cerca de 40% do desenvolvimento falha ou é abandonado, menos de 25% dos sistemas é integrado propriamente ao negócio, e apenas de 10 a 20% atinge seus critérios de sucesso*".

Os problemas observados no mercado de Tecnologia da Informação têm sido atribuídos à complexidade intrínseca do desenvolvimento de sistemas de *software* (BECK, 2000), (LYCETT et al., 2003), (SCHWABER, 2004). Em resposta a esta complexidade crescente, novas metodologias e abordagens de desenvolvimento de *software* estão sendo criadas e adaptadas de forma a aumentar as chances de sucesso dos projetos de desenvolvimento de *software* (ANDREA et al., 2005). Além disso, formou-se a percepção de que os processos de desenvolvimento de *software* precisam submeter-se a mudanças e refinamentos contínuos para que aumentem sua capacidade de lidar com requisitos e expectativas de todos os *stakeholders*. Estas mudanças e refinamentos são feitos através da avaliação e melhoria de processos (MANZONI; PRICE, 2003).

No contexto das abordagens de desenvolvimento de *software*, uma das áreas de estudo que merece destaque é a *engenharia de requisitos*. Os requisitos são o ponto de partida para toda a definição de um sistema de *software* e, conseqüentemente, são fatores decisivos no desenvolvimento do produto final de um projeto de *software*. A engenharia de requisitos é apontada como um dos principais riscos e um dos principais fatores de sucesso de projetos de *software* (BOEHM, 1991), (SOFTWARE ENGINEERING INSTITUTE, 2002), (SOMMERVILLE, 2001).

1.1.2 O Processo de Engenharia de Requisitos Proposto

Considerando-se os resultados dos projetos de desenvolvimento *software*, seus altos índices de insucesso (CLEGG et al., 1996), e a importância da engenharia de requisitos como fator de contribuição para tais resultados, este trabalho propõe um processo de engenharia de requisitos que se adapta ao contexto dos projetos de *software* da atualidade e preocupa-se com a busca de qualidade e melhoria contínua, com o objetivo de aumentar as chances de sucesso dos projetos que o utilizam.

O ponto de partida para a definição do processo proposto neste trabalho é sua preocupação com qualidade e melhoria contínua. As orientações fundamentais deste processo encontram-se na *Integração de Modelos de Capacidade e Maturidade*, ou CMMI: *Capability Maturity Model Integration* (SOFTWARE ENGINEERING INSTITUTE, 2002). Os modelos do CMMI mapeiam processos que estão sendo utilizados em uma organização e identificam pontos de melhoria, contribuindo com o desempenho geral obtido em cada projeto.

Os modelos CMMI foram criados no *Instituto de Engenharia de Software da Universidade de Carnegie Mellon* (SEI/CMU), em Pittsburgh, nos Estados Unidos, e estão sendo adotados como padrão de qualidade no mercado de software atual. Até Julho de 2005, mais de 700 organizações ao redor do mundo já haviam se submetido a avaliações oficiais de conformidade com os modelos CMMI feitas pelo SEI/CMU (SOFTWARE ENGINEERING INSTITUTE, 2005).

Os modelos do CMMI são formados por *áreas de processo*, que agrupam *metas* (*goals*) relacionadas a um determinado contexto. Para que seu processo de desenvolvimento esteja em conformidade com requisitos do CMMI, uma organização deve gradualmente alcançar as metas predefinidas para cada área de processo. Conforme a organização alcança essas metas, diz que está aumentando seu nível de maturidade, que varia de 1 a 5.

Duas áreas de processo do CMMI relacionam-se à engenharia de requisitos: *Gerência de Requisitos*, que faz parte do nível 2 de maturidade e *Desenvolvimento de Requisitos*, relacionada ao nível 3 de maturidade. O processo definido neste trabalho procura alcançar as metas destas duas áreas de processo.

Embora as áreas de processo contenham orientações e descrevam práticas a serem empregadas durante a execução de um processo de desenvolvimento, o CMMI não fornece informações sobre que atividades devem ser executadas em que ordem, quais são as responsabilidades de cada membro da equipe, quais são os produtos de trabalho gerados ao longo do processo. Para definir

estes elementos, este trabalho utiliza, como base, o *Processo Unificado da Rational*, ou RUP – *Rational Unified Process* (KRUCHTEN, 2001).

RUP é um *framework* de processo composto por uma descrição genérica de processos que deve ser adaptada para cada organização e para cada projeto que o utiliza. As atividades do *framework* de processo RUP são agrupadas, por afinidade, em *disciplinas*. A disciplina de *Requisitos* descreve as atividades, papéis e produtos de trabalho relacionados à engenharia de requisitos, e foi utilizada, como um catálogo de elementos de processo, para compor o processo proposto neste trabalho.

A partir destas duas abordagens, CMMI e RUP, foi definido um processo de engenharia de requisitos, para uma organização específica, que tem como principais características: o aproveitamento do processo informal utilizado pela organização, a busca de qualidade e melhoria contínua do processo traduzida no objetivo de satisfazer os níveis 2 e 3 de maturidade do CMMI e a preocupação constante com praticidade, eficiência e retorno do investimento.

Com o objetivo de assegurar os objetivos relacionados a praticidade, eficiência e qualidade, decidiu-se inserir, no processo proposto, melhores práticas de desenvolvimento focadas nestes objetivos. Para isto, foram utilizados *métodos ágeis* (BECK et al., 2001). Estes métodos utilizam um processo de desenvolvimento de software *leve* (*lightweight*), sem uma definição rigorosa de atividades e produtos de trabalho, que valoriza a comunicação e colaboração interpessoal, a geração de resultados tangíveis e a capacidade de acomodação de mudanças.

Assim, define-se a fundamentação teórica do processo de engenharia de requisitos proposto neste trabalho:

- O processo está em conformidade com as áreas de processo do CMMI de Gerência de Requisitos (nível 2) e Desenvolvimento de Requisitos (nível 3).
- Os componentes do processo (atividades, papéis, produtos de trabalho) são baseados no *framework* de processo RUP.
- Práticas de métodos ágeis são recomendadas para utilização durante a execução do processo.

1.1.3 A Institucionalização do Processo

Este trabalho documenta a institucionalização do processo de engenharia de requisitos proposto em uma organização. De fato, a organização citada passou por uma etapa de definição e institucionalização de um processo de desenvolvimento de *software* global, no qual se insere o processo de engenharia de requisitos descrito neste trabalho.

A institucionalização do processo, documentada neste trabalho, envolveu a preparação da organização para receptividade e colaboração com o processo instituído, a liberação dos recursos necessários, a preparação do ambiente técnico e de trabalho e a definição de mecanismos de avaliação.

O método seguido neste trabalho dá ênfase à aplicação das abordagens de desenvolvimento pesquisadas, associando conceitos à sua experimentação real. Assim, ao invés de propor uma integração ampla destas abordagens, restringe o escopo de atuação ao processo de engenharia de requisitos e descreve este processo em detalhes, considerando todas as implicações de

sua institucionalização. As conclusões alcançadas partem do aprendizado adquirido durante a definição, institucionalização e avaliação do processo.

1.2 Trabalhos Relacionados

Em (BOEHM; TURNER, 2003), propõe-se um método para a definição de "*estratégias de desenvolvimento*" baseado na análise de riscos de cada projeto. As abordagens indicadas para compor as estratégias de desenvolvimento são os métodos ágeis, as abordagens classificadas como *guiadas por planos* (com ênfase no planejamento) e métodos baseados em CMMI. O método proposto é focado em estratégias de resolução de riscos, e não na definição de um processo de desenvolvimento, como o processo de engenharia de requisitos proposto neste trabalho.

A abordagem PMT, *Pattern-based Methodology Tailoring*, apresentada em (HARTMANN; PRICE, 2004), também utiliza critérios de análise de risco para instanciar metodologias de desenvolvimento de *software*. Nesta abordagem, os processos de desenvolvimento instanciados são compostos por padrões organizacionais, soluções recorrentes para a organização do trabalho humano que foram observadas em projetos de sucesso comprovado. As características específicas e os riscos associados a cada projeto são utilizados como entrada para a determinação dos padrões organizacionais a serem utilizados para compor o processo de desenvolvimento do projeto. A PMT também não define um processo específico a ser seguido, voltando-se mais para a forma de instanciação dos processos do que para componentes que os integram.

Em (MANZONI; PRICE, 2001-b), são identificadas extensões necessárias ao *framework* RUP para alcançar compatibilidade com CMM. Contudo, nenhuma extensão é recomendada para a área de gerência de requisitos. Além disso, esta análise de compatibilidade considera práticas de CMM e não de CMMI, que é uma das abordagens utilizadas na definição do processo de engenharia de requisitos deste trabalho.

Diversos outros estudos discutem variadas formas de integração das abordagens estudadas neste trabalho – CMMI, RUP e métodos ágeis – em diferentes combinações. Diversos autores investigaram a conformidade de RUP com relação a CMMI, considerando seus princípios e práticas, e definindo possíveis complementações (TYSON; BROWNSWORD; BROWNSWORD, 2004), (GALLAGHER; BROWNSWORD, 2001), (MANZONI; PRICE, 2003), (FITZGERALD; O'KANE, 1999), (SMITH, 2000), (AMBLER, 2001). Alguns estudos analisam a compatibilidade de RUP e métodos ágeis, seus pontos comuns e suas áreas de risco, além de estratégias para processos de desenvolvimento *híbridos* (KRUCHTEN, 2001), (KRUCHTEN, 2001b), (AMBLER, 2001), (AMBLER, 2002), (SANTOS, 2002), (MARTIN, 1998). Por fim, existem estudos que afirmam que CMMI e métodos ágeis podem ser empregados conjuntamente, criando uma sinergia que possibilita que as organizações que os utilizam beneficiem-se das vantagens trazidas por ambos (JEFFRIES, 2000), (REIFER, 2003), (PAULK, 2001). Estes estudos sobre integração de metodologias são descritos brevemente na seção de *Integração de Metodologias de Desenvolvimento de Software*, no capítulo 2, *Fundamentação do Processo*.

Uma abordagem que segue a mesma filosofia de integração de metodologias empregada neste trabalho é descrita em (LYCETT et al, 2003).

Esta abordagem desenvolveu uma abordagem baseada em *framework* de processo para implementar valores e princípios ágeis no contexto de desenvolvimento de uma organização que opera em conformidade com RUP e CMMI. Com foco nos valores, princípios e práticas dos métodos ágeis, determinadas características do RUP foram exploradas de forma a aumentar a produtividade do processo adotado pela organização. Enquanto a abordagem descrita em (LYCETT et al, 2003) concentra-se em estratégias de construção do *framework* de processos da organização, este trabalho concentra-se no processo em si, suas atividades, papéis e artefatos, bem como em sua institucionalização.

1.3 Estrutura do Trabalho

Os capítulos 2 e 3 descrevem a fundamentação teórica deste trabalho. No capítulo 2, *Fundamentação do Processo*, são introduzidos alguns conceitos básicos, utilizados ao longo do texto, e são apresentadas as três abordagens de desenvolvimento utilizadas como orientação na criação do processo de engenharia de requisitos proposto: CMMI, RUP e métodos ágeis. Ao final do capítulo, possibilidades de integração destas abordagens são discutidas e algumas experiências descritas na literatura são analisadas.

O capítulo 3, *Processos e Práticas da Engenharia de Requisitos*, caracteriza a engenharia de requisitos, sua definição e composição, os conceitos, problemas e soluções relacionadas, além de rever as abordagens apresentadas no capítulo 2 sob a ótica da engenharia de requisitos.

O processo proposto neste trabalho é apresentado no capítulo 4, *Processo de Engenharia de Requisitos Proposto*. O processo é contextualizado com relação à organização a que atende e seu processo global de desenvolvimento. A metodologia utilizada na definição do processo é descrita, seguida de uma descrição detalhada da composição do processo, suas atividades, papéis, artefatos e práticas recomendadas.

O capítulo 5, *Institucionalização do Processo*, descreve as etapas envolvidas na institucionalização, o planejamento, a definição de recursos, a preparação do ambiente técnico e cultural e os mecanismos de apoio ao processo.

Por fim, são apresentadas as conclusões do trabalho no capítulo 6.

2 FUNDAMENTAÇÃO DO PROCESSO

O processo de engenharia de requisitos proposto neste trabalho foi criado com base em três abordagens de desenvolvimento:

- A *Integração de Modelos de Capacidade e Maturidade (*CMMI – *Capability Maturity Model Integration*), um modelo de processos utilizado para identificar pontos de melhoria de processos de desenvolvimento de sistemas.
- O *Processo Unificado da Rational Corporation* (RUP – *Rational Unified Process*), um *framework* de processo de desenvolvimento de software que descreve atividade, artefatos e papéis desempenhados durante o desenvolvimento de um projeto desenvolvimento de *software*.
- Os *métodos ágeis*, um conjunto de abordagens de desenvolvimento que compartilham valores e princípios fundamentais e que descrevem boas práticas para o desenvolvimento de sistemas de *software*.

Este capítulo descreve estas três abordagens, bem como diversos estudos feitos no sentido de integrá-las, em processos de desenvolvimento de *software*, com o intuito de obter os benefícios trazidos pela sua utilização.

2.1 Conceitos Básicos

Antes de apresentar detalhes sobre cada abordagem, esta seção estabelece parte da terminologia envolvida na descrição das próximas seções e ao longo deste trabalho, através de definições dos elementos que compõem as abordagens, encontradas na literatura.

2.1.1 Desenvolvimento de Software

2.1.1.1 Processo e Modelo de Processo

As definições a seguir são apresentadas em (SOFTWARE ENGINEERING INSTITUTE, 2003).

Um processo é um conjunto de práticas executadas para atingir determinado objetivo; pode incluir ferramentas, métodos, materiais e pessoas.

Um modelo de processo é uma coleção estruturada de elementos que descreve características de processos efetivos. O CMMI é um modelo de processos cuja utilização é comprovadamente efetiva, através de um histórico amplo de projetos utilizado como base para sua criação.

Modelos de processo são utilizados para definir objetivos e prioridades de melhoria de processo; para contribuir com a garantia de que um processo é estável, capaz e maduro; como orientação para a melhoria de processos de um

projeto ou de uma organização; ou ainda como uma forma de declarar o estado dos esforços de melhoria de uma organização, empregando uma metodologia de avaliação.

É importante notar, ainda, que modelos são abstrações da realidade e, por isso, não devem ser empregados diretamente como foram definidos, mas sim adaptados de acordo com a realidade da organização, o domínio de aplicação, o projeto realizado, etc.

A qualidade de um sistema é altamente influenciada pela qualidade do processo utilizado em sua obtenção, desenvolvimento e manutenção.

2.1.1.2 Metodologia

Uma metodologia é uma coleção recomendada de fases, procedimentos, regras, técnicas, ferramentas, documentação, gerência e treinamento utilizados para desenvolver um sistema. Existe uma filosofia por trás da metodologia, um conjunto de crenças e suposições que a apóiam, explicando por que a metodologia funciona da forma como funciona (AVISON; FITZGERALD, 2003). Esta filosofia pode ser representada por meio de valores, princípio ou práticas fundamentais da metodologia.

2.1.2 Terminologia

2.1.2.1 Stakeholder

O *stakeholder* é qualquer pessoa materialmente afetada pelo resultado do projeto: clientes, usuários diretos e indiretos, investidores, acionistas, fornecedores, supervisores, gerentes, compradores, pessoal de suporte e manutenção, redatores técnicos (que documentam o sistema) (IBM RATIONAL CORPORATION, 2003), (SOFTWARE ENGINEERING INSTITUTE, 2002), (AMBLER, 2002).

Stakeholders pertencem a diversos grupos distintos dentro de uma organização e, assim, apresentam diferentes pontos de vista a respeito do sistema. Alguns estão tão envolvidos com o sistema a ponto de suas carreiras dependerem do mesmo. Outros, são envolvidos apenas indiretamente, pelos resultados de negócio que o sistema influencia. Alguns têm força decisiva sobre o sistema, outros apenas colaboram com informações e experiência.

Algumas definições incluem a equipe de desenvolvimento como possíveis *stakeholders*: projetistas, testadores, programadores, demais desenvolvedores. A literatura relacionada também menciona os conceitos de *stakeholder relevante*, aquele com poder de decisão e cuja colaboração com o projeto é significativa, e *representante de stakeholder*, uma pessoa que substitui o *stakeholder* quando este não está disponível. (IBM RATIONAL CORPORATION, 2003) (SOFTWARE ENGINEERING INSTITUTE, 2002)

A compreensão de quem são os stakeholders e suas necessidades particulares são elementos chave no desenvolvimento de uma solução efetiva de sistema de software (IBM RATIONAL CORPORATION, 2003). *Os stakeholders estão diretamente envolvidos na orientação, forma e escopo do projeto* (RATIONAL UNIVERSITY, 2002-b).

2.1.2.2 Cliente

É um tipo especial de *stakeholder*. Em (SOFTWARE ENGINEERING INSTITUTE, 2002), representa o responsável pelo orçamento do projeto. Neste trabalho, este termo é utilizado para representar os *stakeholders* do projeto que interagem com a equipe de desenvolvimento para definir os requisitos do sistema.

2.1.2.3 Elicitar

Significa extrair, obter, produzir os requisitos do sistema. *Pode utilizar técnicas sistemáticas como protótipos ou entrevistas estruturadas, para identificar e documentar proativamente necessidades dos stakeholders* (SOFTWARE ENGINEERING INSTITUTE, 2002).

2.1.2.4 Cenário

Seqüência de eventos que podem ocorrer durante a utilização de um sistema. Cenários podem ser usados para explicitar necessidades específicas de clientes ou para contribuir com a definição de um caso de uso (SOFTWARE ENGINEERING INSTITUTE, 2002).

2.1.2.5 Disciplina

É um corpo de conhecimento disponível, relacionado a um modelo de processo (SOFTWARE ENGINEERING INSTITUTE, 2002). Agrupa atividades de um processo de desenvolvimento de software, de acordo com sua natureza (KRUCHTEN, 2001).

2.1.2.6 Artefato

É uma porção de informação que é produzida, modificada ou utilizada por um processo. Artefatos são os produtos tangíveis de um projeto: as coisas que o projeto produz ou usa enquanto trabalhando rumo ao produto final. Podem ser modelos como, por exemplo, diagramas de classe; elementos de modelos como as próprias classes; documentos; código fonte; código executável. Também podem ser compostos por outros artefatos (KRUCHTEN, 2001).

2.2 Integração de Modelos de Capacidade e Maturidade (CMMI)

As seções a seguir descrevem o CMMI, sua estrutura e seus objetivos, os conceitos utilizados em sua elaboração, a organização, classificação e composição de seus elementos. A fonte primária das informações apresentadas nestas seções é o relatório técnico que define o CMMI para Engenharia de *Software*, de autoria do SEI/CMU (SOFTWARE ENGINEERING INSTITUTE, 2002).

2.2.1 Definição e Objetivos

Com o objetivo de criar condições para a evolução das boas práticas da engenharia de *software*, o Instituto de Engenharia de Software da Universidade de Carnegie Mellon (SEI/CMU), em Pittsburg, Estados Unidos, promoveu uma série de estudos que resultaram na documentação de um conjunto de melhores

práticas para processos de desenvolvimento de *software*, descritas no *Modelo de Capacidade e Maturidade*, o CMM (*Capability Maturity Model*).

2.2.1.1 Modelos de Processo Integrados

Dependendo da área de aplicação, diferentes modelos foram criados, tais como *Software-CMM* (CMM para *Software*) e *SA-CMM* (CMM para Aquisição de Software). O CMMI (*Capability Maturity Model Integration*), ou *Integração de Modelos de Capacidade e Maturidade*, representa a integração e evolução destes modelos em um volume único, que deverá substituir os múltiplos modelos de CMM previamente definidos. As pesquisas e todas as atividades envolvidas no estudo destes modelos pelo SEI possuem o apoio do Departamento de Defesa dos Estados Unidos.

2.2.1.2 Objetivos

O objetivo do CMMI é melhorar os processos da organização que o adota, assim como sua habilidade de gerenciar o desenvolvimento, aquisição e manutenção de produtos ou serviços. Neste sentido, o CMMI busca servir como uma estruturação de práticas e abordagens de sucesso comprovado. Tais práticas visam contribuir com a organização através da disponibilização de orientações sobre: como avaliar sua maturidade organizacional e sua capacidade por área de processo; como estabelecer prioridades para melhora; e como implementar tal melhora. O CMMI é formado por um pacote de produtos que reúne múltiplos modelos de processo, associados a seus manuais, material de treinamento e de avaliação.

Os modelos que integram o CMMI têm o objetivo de integrar gerenciamento de qualidade, melhores práticas aplicadas a determinados domínios e práticas de mudança da organização (TYSON; BROWNSWORD; BROWNSWORD, 2004).

Em (SMITH, 2000), afirma-se que o CMM fornece um mecanismo de avaliação bem estabelecido de maturidade de processo. Ainda segundo Smith, o CMM se tornou um veículo popular para a determinação da maturidade do processo de desenvolvimento de *software* de organizações em diversos domínios.

2.2.1.3 Representações e Corpos de Conhecimento

Os modelos CMMI variam de acordo com a representação e com o corpo de conhecimento (*body of knowledge*) escolhido pela organização. Existem duas representações: *contínua* ou *em estágios*. A representação contínua permite que se escolha a ordem de implementação das áreas de processo. Já a representação em estágios define uma seqüência comprovada de melhorias, começando com práticas de gerência e progredindo ao longo de um caminho com níveis sucessivos predefinidos, cada um servindo como base para o próximo.

Quanto aos corpos de conhecimento, estes representam domínios de aplicação, cada um com seu modelo de processo correspondente. Os modelos definidos no CMMI atualmente correspondem aos seguintes corpos de conhecimento, também chamados de *disciplinas*:

- *Engenharia de Sistemas*: práticas relacionadas ao desenvolvimento completo de sistemas (de *software* ou qualquer outro tipo de sistema);
- *Engenharia de Software*: práticas relacionadas ao desenvolvimento de sistemas de *software*;
- *Desenvolvimento Integrado de Produto e Processo (IPPD)*: abordagem sistemática relacionada à colaboração dos envolvidos.
- *Fornecedor Externo*: práticas relacionadas à subcontratação de fornecedores.

Está prevista, ainda, a possibilidade de inclusão de novas disciplinas no CMMI ao longo do tempo. **Este trabalho mantém o foco na representação do CMMI em estágios, e na disciplina de Engenharia de *Software*.**

2.2.1.4 Aplicação em Organizações

A adoção ou implantação de um modelo CMMI deve ser feita observando-se as particularidades da organização, do ambiente de negócios e das demais circunstâncias envolvidas. Assim, evidencia-se o conceito do *framework* CMMI, que representa um conjunto completo de modelos que será customizado de acordo com as necessidades da organização previamente à sua adoção.

Antes de aplicar um modelo CMMI para melhorar os processos de uma organização, é preciso mapear os processos da organização para áreas de processo (PAs) do CMMI. Este mapeamento permite determinar o nível de conformidade ao modelo CMMI atingido pela organização. Nem todos os processos da organização terão um mapeamento direto para uma área de processo CMMI, já que as metas de uma PA podem estar sendo satisfeitas pelo conjunto de processos da organização.

2.2.2 Conceitos Utilizados

Alguns conceitos básicos utilizados na construção de modelos CMMI, ilustrados na figura a seguir, são:

- *Nível de maturidade*: representa um estágio evolucionário definido, que pode ser atingido por uma organização que utiliza elementos dos modelos CMMI de forma completa ou parcial. Os níveis são uma forma de priorizar as ações de melhoria de tal forma que se aumente a maturidade do processo de *software*. Por exemplo, no nível 2 são tratados aspectos gerenciais dos projetos, e no nível 3 são tratados aspectos técnicos de desenvolvimento (MANZONI; PRICE, 2001).
- *Capacidade de Processo*: descreve o conjunto de resultados esperados e que podem ser alcançados, por seguir o processo de software. A capacidade de processo de software de uma organização provê os meios de predizer os resultados esperados para o próximo projeto da organização (PAULK, 1993 apud MANZONI; PRICE, 2001)
- *Áreas de Processo (Process Areas)*: uma área de processo é o agrupamento de práticas relacionadas a determinado contexto que, quando executadas de forma coletiva, satisfazem uma série de metas consideradas

importantes para alcançar uma melhora significativa naquele contexto (ou seja, atingir um certo *nível de maturidade*).

- *Metas Específicas*: descrevem características a ser implementadas para satisfazer uma área de processo.
- *Práticas Específicas*: atividades consideradas importantes para alcançar uma determinada meta específica.
- *Subpráticas*: descrições detalhadas que orientam a interpretação de práticas específicas ou genéricas.
- *Metas Genéricas*: são rotuladas de “genéricas” porque podem aparecer em mais de uma área de processo. Atingir uma meta genérica em uma área de processo significa obter uma melhora no controle do planejamento e da implementação de processos associados àquela área de processo. Essa melhora indica que tais processos são provavelmente efetivos, passíveis de repetição e duradouros.
- *Práticas Genéricas*: boas práticas a ser executadas com o objetivo de alcançar as metas genéricas. Fornecem orientações para que os processos associados a uma área de processo sejam efetivos, passíveis de repetição e duradouros. As práticas genéricas são classificadas em diferentes categorias, chamadas de *características comuns*.
- *Características Comuns*: agrupamentos utilizados para apresentar práticas genéricas. Existem apenas na representação por estágios (níveis de maturidade) do CMMI. As características comuns são:

  – *Compromisso com a Execução*: agrupa práticas relacionadas à criação de políticas organizacionais e à garantia de apoio ao processo por parte dos diversos membros da organização, sobretudo a direção;

  – *Habilidade para a Execução*: as práticas desta característica garantem a disponibilidade dos recursos necessários para a execução do processo;

  – *Direcionamento da Implementação*: agrupa práticas genéricas que orientam a execução do processo através do controle de seu desempenho, da gerência da integridade de produtos de trabalho, e do envolvimento de *stakeholders* relevantes;

  – *Verificação da Implementação*: categoria de práticas relacionadas à revisão pela gerência sênior e à avaliação objetiva de aderência ao processo definido pela organização.
- *Produtos de Trabalho*: resultados de saída de alguma atividade: artefatos, documentos, planilhas, listas, avaliações, gráficos, etc.
- *Institucionalização*: este conceito representa a definição, promoção, adoção e avaliação de processos definidos ao longo de toda a organização. Institucionalizar um processo significa criar políticas organizacionais para tal processo, planejar sua utilização, divulgar este planejamento, criar formas de estimulação do uso do processo, quantificar seus resultados e criar bases históricas em um repositório para futuras consultas. Para cada nível de maturidade, a institucionalização de processos possui características e requisitos próprios.

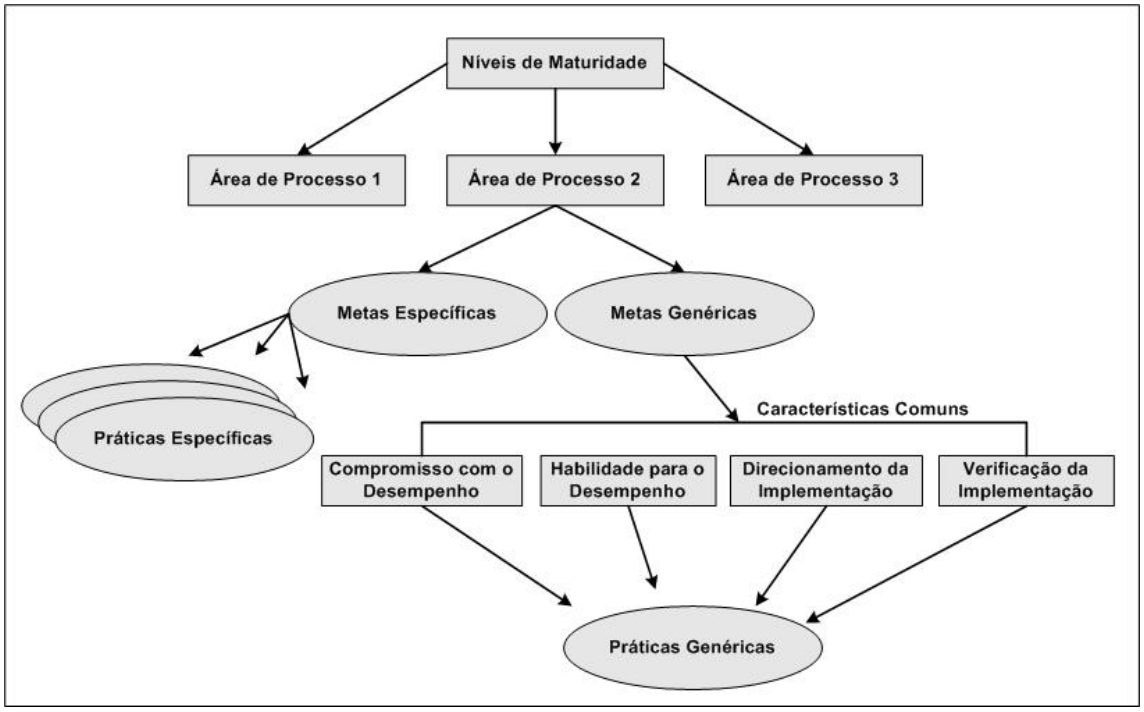


Figura 2.1: Componentes de modelos CMMI (SOFTWARE ENGINEERING INSTITUTE, 2002)

2.2.3 Áreas de Processo

Uma área de processo é um conjunto de práticas relacionadas a determinado contexto que, quando executadas de forma coletiva, satisfazem uma série de objetivos considerados importantes para alcançar uma melhora significativa naquele contexto.

Há 4 (quatro) categorias de área de processo nos modelos CMMI: *Gerenciamento de Processo*, *Gerenciamento de Projeto*, *Engenharia* e *Suporte*. A tabela abaixo apresenta estas categorias associadas às áreas de processo que as compõem. Note-se que as áreas de processo discutidas neste trabalho não incluem aquelas relacionadas ao modelo IPPD (*Desenvolvimento Integrado de Produto e Processo*) ou à Gerência de Fornecedores, já que o foco deste trabalho são os modelos relacionados à Engenharia de *Software*.

Tabela 2.1: Áreas de Processo do CMMI por Categoria

Categoria	Área de Processo
Gerenciamento de Processo	Foco no Processo Organizacional (*Organizational Process Focus*)
	Definição do Processo Organizacional (*Organizational Process Definition*)
	Treinamento da Organização (*Organizational Training*)
	Desempenho de Processo Organizacional (*Organizational Process Performance*)

	Implementação e Inovação Organizacional (*Organizational Innovation and Deployment*)
Gerenciamento de Projeto	Planejamento de Projeto (*Project Planning*)
	Monitoramento e Controle de Projeto (*Project Monitoring and Control*)
	Gerência de Acordo com Fornecedor (*Supplier Agreement Management*)
	Gerência de Riscos (*Risk Management*)
	Gerência Quantitativa de Projeto (*Quantitative Project Management*)
Engenharia	Desenvolvimento de Requisitos (*Requirements Development*)
	Gerência de Requisitos (*Requirements Management*)
	Solução Técnica (*Technical Solution*)
	Integração do Produto (*Product Integration*)
	Verificação (*Verification*)
	Validação (*Validation*)
Suporte	Análise e Medida (*Measurement and Analysis*)
	Garantia de Qualidade de Processo e de Produto (*Process and Product Quality Assurance*)
	Gerência de Configuração (*Configuration Management*)
	Análise e Resolução de Causas (*Causal Analysis and Resolution*)
	Análise e Resolução de Decisão (*Decision Analysis and Resolution*)

2.2.4 CMMI em Estágios e os Níveis de Maturidade

Conforme foi descrito anteriormente, a representação do CMMI em estágios define uma seqüência comprovada de melhorias, progredindo ao longo de um caminho com níveis sucessivos de maturidade organizacional. Cada nível serve como base para o próximo, de forma a priorizar as ações de melhoria conforme o que se observou ser uma ordem eficiente de execução dessas ações. Os níveis sucessivos de maturidade são estágios evolucionários que variam do 1 (*Inicial*) ao 5 (*Em Otimização*).

Conforme ilustrado na Figura 2.1, cada nível de maturidade trabalha determinado grupo de áreas de processo. Para cada área de processo, são definidas metas e práticas específicas, seguidas de metas e práticas genéricas. Uma organização atinge um determinado nível de maturidade quando alcança as metas e emprega as técnicas estipuladas para as áreas de processo daquele nível.

Os 5 níveis de maturidade do CMMI são descritos nas seções a seguir (ver Figura 2.2). O processo de engenharia de requisitos proposto neste trabalho se insere em um processo global de desenvolvimento de *software* que visa atingir o nível 3 de maturidade organizacional. Assim, no que se refere às áreas de processo relacionadas a requisitos, a organização deve ter as características descritas a seguir correspondentes aos níveis 2 (*Gerenciado*) e 3 (*Definido*).

2.2.4.1 Nível de Maturidade 1: Inicial

Neste nível se enquadram organizações que não possuem um processo definido – também considerado o nível do *caos*. Neste caso, os resultados de cada projeto são imprevisíveis e em geral só são positivos quando os envolvidos apresentam desempenho classificado em (SOFTWARE ENGINEERING INSTITUTE, 2002) como *heróico*.

Outras características comuns destas organizações são a tendência a criar compromissos que não conseguem cumprir, o estouro de orçamento e a incapacidade de reproduzir resultados bem sucedidos.

2.2.4.2 Nível de Maturidade 2: Gerenciado

Em organizações deste nível de maturidade, os projetos possuem um gerenciamento satisfatório de requisitos e os processos utilizados são planejados, executados, mensurados e controlados.

Diferente do que acontece no nível de maturidade *Inicial*, em organizações de nível *Gerenciado* os processos definidos não são abandonados em momentos de crise. Requisitos, processos, produtos e serviços são gerenciados e seu *status* é visível para a gerência em pontos de checagem claramente definidos (*marcos* ou *milestones*).

2.2.4.3 Nível de Maturidade 3: Definido

Neste estágio, além dos processos estarem bem definidos e entendidos, há uma descrição formal destes envolvendo padrões, procedimentos, ferramentas e métodos. Os processos são estabelecidos e melhorados ao longo do tempo.

Cada projeto utiliza a descrição padrão de processos da organização como um *framework*, e o configura com o objetivo de adaptar o processo padrão às características do projeto. A configuração é feita seguindo regras definidas no processo padrão, o que garante que diferentes projetos não apresentem variações muito grandes entre si, aumentando, assim, o controle sobre os resultados.

2.2.4.4 Nível de Maturidade 4: Gerenciado Quantitativamente

Partindo da descrição padrão de processos da organização caracterizada no item anterior, organizações com nível 4 de maturidade selecionam subprocessos chave para melhoria de desempenho nos resultados dos projetos. Estes subprocessos são então controlados através de técnicas estatísticas e quantitativas.

O controle destes subprocessos é feito da seguinte forma:

- São definidos objetivos quantitativos para qualidade e desempenho;

- Durante a execução do subprocesso, seus resultados são medidos através das técnicas mencionadas anteriormente;
- Processos de gerenciamento utilizam-se dos objetivos e das medidas durante o andamento do subprocesso para assegurar resultados positivos (pode-se identificar, por exemplo, causas de variações nos índices de desempenho e então corrigi-las).

Medidas de qualidade e desempenho de processos são armazenadas no repositório corporativo para sedimentar futuras decisões.

2.2.4.5 Nível de Maturidade 5: Em Otimização

Neste nível de maturidade, os processos da organização são continuamente melhorados com base na interpretação quantitativa das causas comuns de variação inerentes aos processos. Essas melhorias são feitas com o objetivo de alcançar metas definidas pela empresa e constantemente revisadas.

A habilidade da organização de responder a mudanças e oportunidades rapidamente é alavancada pela utilização de formas de acelerar e compartilhar o conhecimento. Todos os participantes estão envolvidos na melhoria de processos.

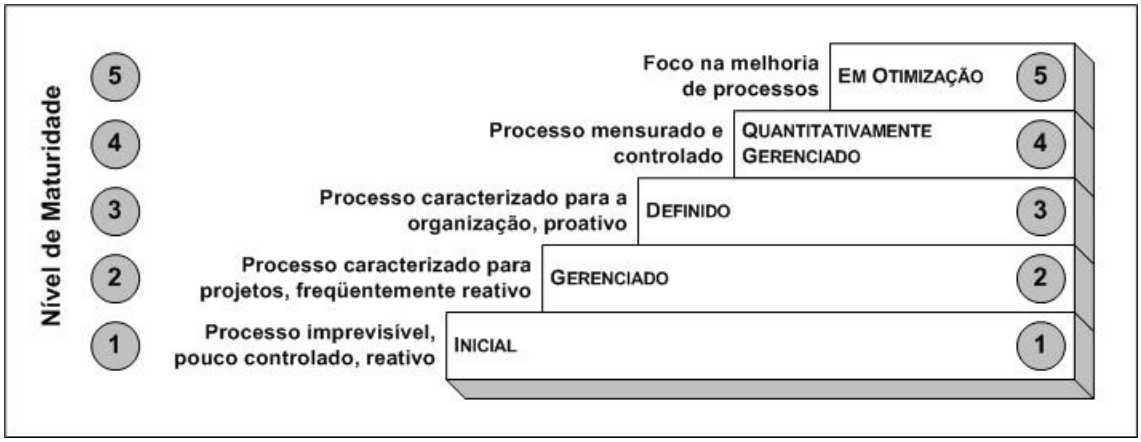


Figura 2.2: Níveis de Maturidade do CMMI [adaptado de (REITZIG, 2003)]

2.2.5 Nível de Maturidade 2: Gerenciado

Esta seção descreve os requisitos que classificam uma organização como pertencente ao nível 2 de maturidade do modelo CMMI.

2.2.5.1 Áreas de Processo

Conforme foi mencionado anteriormente, cada nível de maturidade trabalha determinadas áreas de processo. As áreas de processo consideradas para o nível 2 de maturidade do CMMI são:

- *Gerência de Requisitos* (*REQM – Requirements Management*): É responsável pela manutenção dos requisitos. Descreve atividades para manter e controlar mudanças nos requisitos e para fazer com que outros dados e planos relevantes estejam sempre atualizados. Provê rastreabilidade de requisitos do cliente ao produto e aos componentes de produtos. Como as mudanças em requisitos podem afetar todas as outras áreas da Engenharia, a área de processo de Gerência de Requisitos é uma seqüência de eventos dinâmica e geralmente recursiva.

- *Planejamento de Projeto* (*PP – Project Planning*): Inclui o desenvolvimento de um plano de projeto, o envolvimento dos *stakeholders* do projeto da forma apropriada, obtenção de compromisso com o plano de projeto e manutenção do plano. O planejamento inicia com os requisitos que definem o produto e o projeto. O plano de projeto cobre as diversas atividades de gerenciamento e engenharia que serão executadas no projeto. Ao definir seu plano específico, um projeto leva em conta outros planos que também o afetam, tais como planos de avaliação de processos, de avaliação de produto, de gerenciamento de configuração e de análise e medida.
- *Monitoramento e Controle de Projeto* (*PMC – Project Monitoring and Control*): Inclui o monitoramento de atividades e a tomada de ações corretivas. O plano de projeto especifica o nível apropriado de monitoramento, a freqüência das revisões de progresso e as medidas a serem usadas para monitorar o progresso. Quando o status corrente do processo apresenta diferenças significativas com relação ao estimado no plano, as ações corretivas apropriadas são tomadas (o que pode incluir re-planejamento).
- *Gerência de Acordo com Fornecedor* (*SAM – Supplier Agreement Management*): Esta área endereça as necessidades de um projeto de adquirir partes de trabalho produzidas por fornecedores. Uma vez que um componente de produto é identificado e seu fornecedor selecionado, um acordo com o fornecedor é estabelecido e mantido. Este acordo serve para gerenciar a interação com o fornecedor. O progresso e o desempenho do fornecedor são gerenciados. Testes e revisões de aceitação são conduzidos no componente adquirido.
- *Análise e Medida* (*MA – Measurement and Analysis*): Esta área de processo dá suporte a todas as outras áreas fornecendo práticas específicas que orientam projetos e organizações no alinhamento das necessidades e objetivos de medição. Isso é feito utilizando-se uma abordagem que forneça resultados objetivamente mensuráveis que possam ser usados para a tomada de decisões e de ações corretivas apropriadas.
- *Garantia de Qualidade de Processo e de Produto* (*PPQA – Process and Product Quality Assurance*): Responsável por fornecer práticas específicas para avaliar objetivamente o desempenho de processos, produtos de trabalho e serviços, comparando-os com as correspondentes descrições de processo, padrões e procedimentos. Caso essa comparação evidencie algum problema, esta área de processo também é responsável por garantir que esses problemas sejam endereçados.
- *Gerência de Configuração* (*CM – Configuration Management*): Responsável por estabelecer e manter a integridade de produtos de trabalho utilizando recursos como identificação de configuração, controle de configuração, relatórios de status de configuração e auditorias. Os produtos de trabalho colocados sob a gerência de configuração incluem produtos que são entregues ao cliente, produtos de trabalho internos selecionados, produtos adquiridos, ferramentas e outros itens utilizados na criação e descrição dos produtos de trabalho mencionados – planos,

descrições de processos, requisitos, dados de projetos, desenhos, especificação de produtos, código, compiladores, arquivos de dados e publicações técnicas sobre produtos.

Cada uma das áreas de processo possui metas específicas e define práticas comprovadas para alcançar estas metas. Este trabalho mantém o foco nas áreas de processo de engenharia de requisitos. Assim, dentre as áreas de processo do nível 2, o interesse está na Gerência de Requisitos, que está descrita em detalhes no capítulo sobre *Processos de Práticas da Engenharia de Requisitos* (ver seção *Engenharia de Requisitos no CMMI*).

2.2.5.2 Meta Genérica – (GG2) Institucionalizar um Processo Gerenciado

Para atingir os requisitos do nível de maturidade 2 do CMMI, ou seja, *Processo Gerenciado*, todas as áreas de processo da organização têm a meta genérica de institucionalizar um processo gerenciado, referenciado pela sigla *GG2* (*Generic Goal 2*).

Para atingir esta meta genérica, é necessário pôr em prática uma série de ações (práticas genéricas) que resultam na institucionalização do processo. As práticas genéricas associadas a esta meta são listadas abaixo:

- *Estabelecer uma política organizacional*: Aderir a políticas organizacionais;
- *Planejar o processo*: Seguir descrições de planos e processos estabelecidos;
- *Fornecer recursos*: Prover recursos adequados (incluindo recursos financeiros, pessoal e ferramentas) para a execução do processo adotado;
- *Designar responsabilidade*: Designar responsabilidade e autoridade para executar o processo;
- *Treinar pessoal*: Treinar o pessoal que executa e dá suporte ao processo;
- *Gerenciar configurações*: Colocar produtos de trabalho selecionados sob níveis apropriados de gerenciamento de configuração;
- *Identificar e envolver stakeholders relevantes*: Identificar e envolver indivíduos chave para o desenrolar do processo de desenvolvimento;
- *Monitorar e controlar o processo*: Monitorar e controlar o desempenho do processo comparando os resultados obtidos com os planejados e tomando ações corretivas quando necessário;
- *Avaliar aderência objetivamente*: Avaliar o processo objetivamente, verificando a aderência de produtos de trabalho e de serviços às descrições de processo, objetivos e padrões e resolvendo problemas de não conformidade;
- *Revisar status com o alto escalão da gerência*: Revisar as atividades, status e resultados do processo com o alto escalão de gerenciamento e tomar ações corretivas.

2.2.6 Nível de Maturidade 3: Definido

A seguir, estão descritas as características de uma organização com maturidade de nível 3 do CMMI. De acordo com a descrição de nível 3

apresentada anteriormente, organizações neste estágio possuem uma descrição formal de seus processos, que podem ser adaptados por cada projeto que os utiliza.

2.2.6.1 Áreas de Processo

Organizações com nível de maturidade *Definido* alcançaram as metas das seguintes áreas de processo:

- *Desenvolvimento de Requisitos* (*RD – Requirements Development*): Identifica necessidades do cliente a as traduz em requisitos do produto. O conjunto de requisitos do produto é analisado para produzir uma solução conceitual de alto nível. Após isso, o conjunto de requisitos é alocado em um conjunto de requisitos de componentes de produto. Outros requisitos que ajudam a definir o produto são então identificados e também traduzidos em requisitos de componentes de produto. Os requisitos descrevem o produto em termos de funcionalidade, desempenho, características, requisitos de verificação, e outros, de forma que o desenvolvedor compreenda sua utilização.
- *Solução Técnica* (*TS – Technical Solurion*): Parte dos requisitos do produto, identificados pela área de processo de Desenvolvimento de Requisitos, para convertê-los na arquitetura do produto, no projeto de seus componentes, e no próprio componente (através de codificação, fabricação ou aquisição). Uma das atividades desta área é a investigação de soluções alternativas com o objetivo de selecionar o melhor projeto (*design*) com base em critérios estabelecidos. Tais critérios dependem do tipo de produto, ambiente operacional, requisitos de performance e de suporte, e cronogramas de entrega e de liberação de recursos.
- *Integração do Produto* (*PI – Product Integration*): A partir dos requisitos do produto, identificados pela área de processo de Desenvolvimento de Requisitos, e dos componentes em si, gerados pela área de Solução Técnica, combina os componentes e certifica-se de que as interfaces satisfazem os requisitos. Para realizar esta integração, estuda-se a melhor seqüência das atividades de integração e de entrega do produto para o cliente.
- *Verificação* (*VER – Verification*): Assegura que os produtos de trabalho selecionados alcançam os requisitos especificados. Inclui a seleção de métodos de verificação e configura um processo incremental, iniciando com a verificação de componentes do produto e culminando na verificação de todos os componentes e produtos completamente montados e integrados. As revisões por pares (*peer reviews*), onde um profissional revisa o trabalho de outro profissional de perfil semelhante, também fazem parte desta área de processo. Este tipo de verificação é um método comprovado para a remoção de defeitos em estágios iniciais do desenvolvimento.
- *Validação* (*VAL – Validation*): Valida o produto, comparando-o às necessidades do cliente, de forma incremental. A coordenação com o cliente na validação de requisitos é um dos elementos essenciais desta área de processo. O escopo da área de processo de Validação inclui a validação de produtos, de componentes de produtos, de produtos de

trabalho intermediários selecionados e de processos. Tais validações podem incluir a necessidade de re-validação (e inclusive de re-verificação). Problemas encontrados durante a validação podem incorrer em alterações nas áreas de Desenvolvimento de Requisitos e Solução Técnica.

- *Foco no Processo Organizacional* (*OPF – Organizatinoal Process Focus*): Ajuda a organização a planejar e implementar melhorias de processo organizacional baseado na compreensão de seus dos atuais pontos fortes e das atuais fraquezas de seus processos e recursos. Há vários meios de identificar possíveis melhorias nos processos organizacionais tais como propostas de melhoria de processo, medições dos processos, lições aprendidas durante a implementação dos processos e resultados de atividades de avaliação dos mesmos.
- *Definição do Processo Organizacional* (*OPD – Organizatinoal Process Definition*): Estabelece e mantém o conjunto de processos padrão e outros itens relevantes relacionados ao processo da organização tais como descrições e elementos de processo, descrições de modelos de ciclo de vida, orientações (*guidelines*) para a customização de processos, demais documentações e dados relacionados a processos, experiências relatadas e produtos de trabalho, dados de medição de processo coletados durante sua execução, artefatos e lições aprendidas. Tais itens são baseados nas necessidades e objetivos do processo organizacional.
- *Treinamento da Organização* (*OT – Organizational Training*): Identifica necessidades de treinamento estratégico da organização e necessidades de treinamento comuns em vários projetos e grupos de apoio. Também é organizado treinamento destinado a desenvolver habilidades necessárias para executar os processos padrão da organização. Fazem parte das responsabilidades desta área de processo itens como a definição de um programa gerenciado de treinamento e a preparação de pessoal com conhecimento e mecanismos apropriados para medir a efetividade do programa de treinamento.
- *Gerência de Riscos* (*RSKM – Risk Management*): Embora a identificação e o monitoramento de riscos sejam cobertos nas áreas de processo de Planejamento de Projeto e Controle e Monitoramento de Projeto, a área de processo de Gerenciamento de Riscos toma uma abordagem mais contínua e com visão de mais longo prazo para gerenciar riscos com atividades que incluem a identificação de parâmetros de risco, a estimativa e a avaliação de riscos e resolução dos mesmos.
- *Análise e Resolução de Decisão* (*DAR – Decision Analysis and Resolution*): Fornece um processo de avaliação formal que garante a comparação de alternativas em busca da melhor escolha para alcançar os objetivos de todas as outras áreas de processo.

No contexto de interesse deste trabalho, a área de processo de Desenvolvimento de Requisitos será caracterizada e analisada em detalhes no capítulo sobre *Processos de Práticas da Engenharia de Requisitos* (ver seção *Engenharia de Requisitos no CMMI*).

2.2.6.2 Meta Genérica – (GG3) Institucionalizar um Processo Gerenciado

Uma organização com o processo de desenvolvimento gerenciado adota práticas que estabelecem a descrição de um processo definido para os projetos e unidades organizacionais e mantém uma coleção de produtos de trabalho, medidas e informações para melhoria derivadas do planejamento e execução de um processo definido. As práticas genéricas associadas à meta de *Institucionalizar um Processo Gerenciado* são:

- *Estabelecer um Processo Definido*: elaborar e manter a descrição de um processo em nível organizacional, que pode ser adaptado em diferentes instâncias, de acordo com características específicas dos projetos. A definição de um processo padrão faz com que os artefatos, os dados e a aprendizagem de um projeto possam ser reutilizados (ainda que parcialmente) em outros projetos e diminui a variação no desempenho dos projetos.
- *Coletar Informações de Melhoria*:os produtos de trabalho resultantes da execução do processo são armazenados para que possam ser reutilizados nos próximos projetos, tanto na execução como no planejamento de atividades.

2.3 O Processo Unificado da Rational (RUP)

A seção anterior descreve, em linhas gerais, o modelo de processos do CMMI, estrutura, áreas de processo, níveis de maturidade e metas genéricas. Basicamente, o modelo do CMMI é definido sobre metas genéricas e específicas, que descrevem os objetivos do processo, e práticas recomendadas para alcançar cada uma das metas. As metas representam filosofias encontradas em projetos de software de sucesso comprovado ao longo do tempo.

Quando uma organização decide adotar o *modelo de processos* do CMMI, ainda assim precisa definir o *processo de desenvolvimento* a ser utilizado. Ou seja, ao adotar a filosofia pretendida, é preciso organizar as práticas recomendadas definindo atividades a serem executadas, artefatos a serem gerados e papéis a serem desempenhados. O CMMI fornece orientação através de metas e práticas, mas o processo define *quem faz o que, como e quando*.

O RUP pode ser utilizado para suprir esta necessidade de definição do processo de desenvolvimento. RUP descreve:

- *Papéis* ou perfis de trabalho que definem *quem* é responsável por cada tarefa;
- *Artefatos* que representam os produtos do processo, *o que* será gerado durante o processo de desenvolvimento;
- *Atividades* executadas durante o processo de desenvolvimentos, que descrevem *como* os representantes de cada papel trabalham para atingir os objetivos do projeto e construir o sistema de software;
- *Fluxos de atividades*, que orientam a execução das atividades, definindo *quando* esta deve ocorrer.

Os elementos descritos acima são organizados em um *framework* de processo de desenvolvimento de *software*. Isto significa que RUP define um

catálogo de elementos de processo que deve ser *instanciado*. A instanciação é a escolha dos elementos que serão utilizados para compor o processo de desenvolvimento a ser utilizado em um projeto específico. Esta escolha é feita a partir das características da organização e do projeto onde o processo será utilizado.

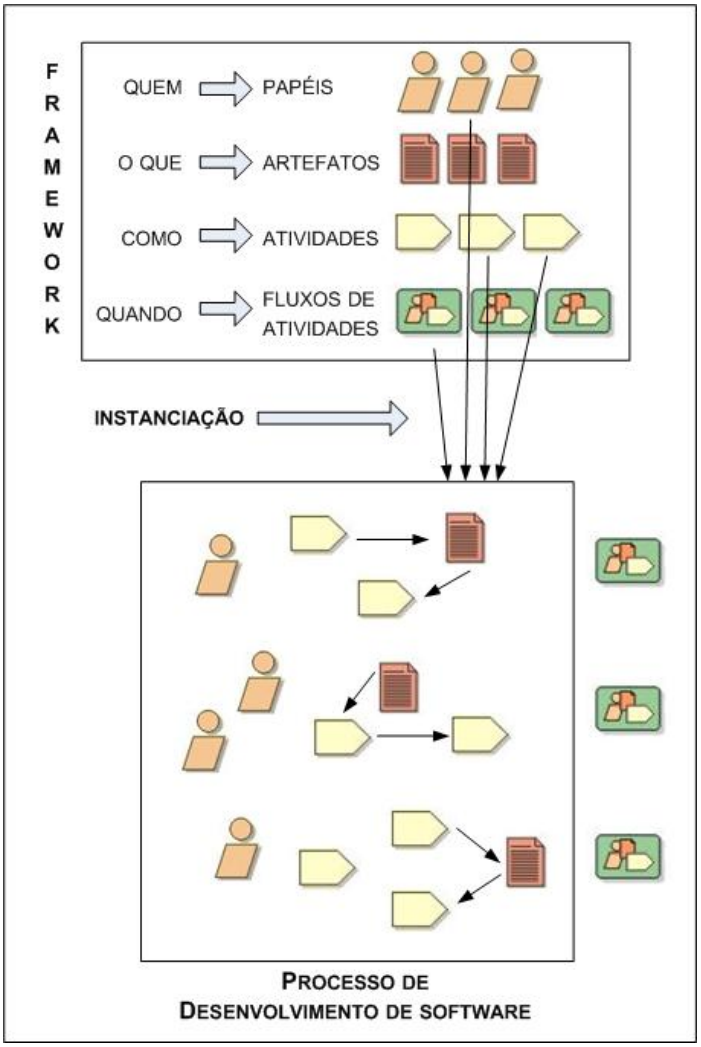


Figura 2.3: Representação do *Framework* de Processo e uma Instância

RUP foi intencionalmente desenvolvido para uma ampla aplicabilidade, ajustando-se a variações de tamanho do projeto; domínio da aplicação (sistemas de negócio, sistemas técnicos); tecnologia utilizada (linguagens, plataformas); e contexto de negócio (desenvolvimento interno, desenvolvimento de produto, contratos de desenvolvimento) (KRUCHTEN, 2001).

RUP é baseado em um modelo de processos que, à semelhança do CMMI, reconhece práticas observadas em projetos de sucesso comprovado. A identificação destas práticas é resultado do trabalho feito, por mais de 10 anos, pelos consultores da empresa originalmente chamada *Rational Software Corporation*, agora integrada à IBM formando a *IBM Rational*.

As seções a seguir descrevem diversos aspectos do RUP, caracterizando sua metodologia de desenvolvimento. As principais fontes consultadas para essa caracterização são o livro *The Rational Unified Process, An Introduction*, escrito por Philippe Kruchten (KRUCHTEN, 2001), o arquiteto líder responsável

por RUP, e a própria ferramenta comercial RUP, em sua versão de 2001 (IBM RATIONAL CORPORATION, 2003).

2.3.1 Práticas de RUP

As práticas recomendadas por RUP foram definidas de forma a aumentar as chances de sucesso de um projeto. Durante a definição do processo RUP, foram identificados alguns sintomas comuns e recorrentes da falha de projetos: entendimento impreciso das necessidades do usuário; falta de habilidade para lidar com mudanças nos requisitos; problemas de integração em módulos do sistema; dificuldade de manutenção e de evolução; descoberta tardia de sérias falhas do projeto; má qualidade do *software*; performance inaceitável do *software*; falta de rastreabilidade, no sentido de saber que membros da equipe alteraram que parte do sistema; processo de liberação de versões de *software* não confiável.

Com base nos sintomas acima, procurou-se descobrir as razões desses problemas, chegando-se à lista de causas a seguir:

- Falta de uma definição formal de um processo de gerenciamento de mudanças;
- Propagação de mudanças descontrolada;
- Comunicação ambígua e imprecisa;
- Arquiteturas não robustas;
- Alta complexidade;
- Inconsistências não detectadas em requisitos, no projeto (*design*) e na implementação;
- Testes insuficientes do sistema;
- Controle subjetivo do status do projeto;
- Falha ao atacar riscos do projeto;
- Automação insuficiente.

A lista acima representa os principais fatores endereçados pelos autores de RUP com a definição das melhores práticas desta metodologia. Outras abordagens de desenvolvimento, como o CMMI ou os métodos ágeis, ambos discutidos neste trabalho, são baseadas em conclusões que apresentam algumas diferenças com relação à percepção de causas de problemas em projetos, embora compartilhem alguns pontos em comum.

De qualquer forma, as práticas recomendadas por RUP foram identificadas em um grande número de projetos de sucesso e sua falta foi verificada em um grande número de projetos fracassados. Estas práticas são ilustradas na figura abaixo, e detalhadas a seguir.

Figura 2.4: Práticas de RUP

2.3.1.1 Desenvolver software iterativamente

O ciclo de vida proposto por RUP é baseado no modelo em espiral (BOEHM, 1985) e consiste em dividir a construção do *software* em iterações, cada uma resultando em uma liberação de versão executável do sistema. Esta abordagem resulta em descoberta, criação e implementação contínuas de requisitos.

O desenvolvimento de *software* de forma iterativa soluciona ou minimiza vários dos problemas de desenvolvimento. Os usuários recebem as liberações de versão de cada iteração, o que facilita seu retorno sobre o *software* produzido, apontando o que foi mal entendido e esclarecendo os demais requisitos. Além disso, os usuários *stakeholders* têm evidências concretas do andamento do sistema. As porções do sistema que oferecem maior risco são desenvolvidas primeiro, e possíveis problemas já podem começar a ser solucionados. O status do projeto é determinado objetivamente, analisando os artefatos que foram concluídos até uma determinada data. A carga de trabalho da equipe é distribuída de maneira uniforme ao longo do projeto. Por fim, a equipe pode utilizar as lições aprendidas em cada iteração para melhorar o processo de desenvolvimento continuamente.

O ciclo de vida iterativo de RUP é ilustrado na figura abaixo.

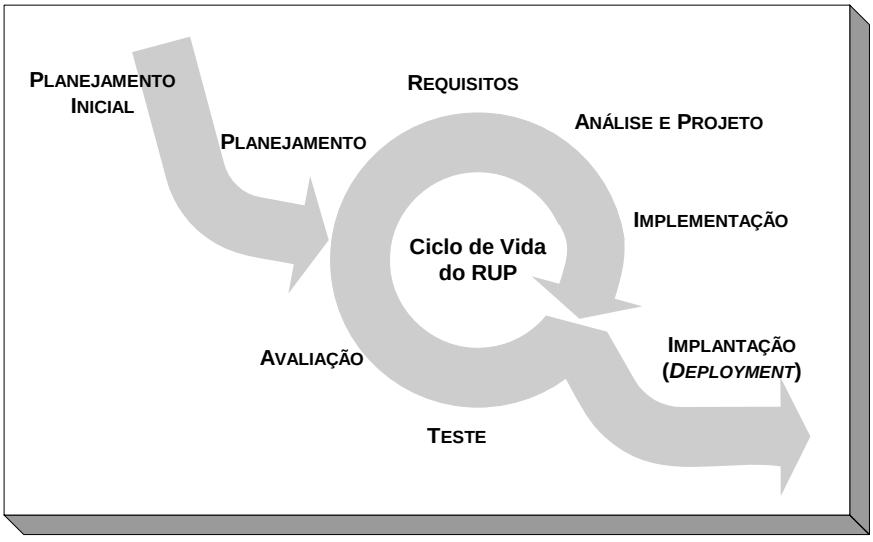


Figura 2.5: Ciclo de Vida de RUP [Adaptada de (KRUCHTEN, 2001)]

2.3.1.2 Gerenciar os requisitos

A gerência de requisitos é sempre uma atividade desafiadora no desenvolvimento de *software*. Isto ocorre porque os requisitos de um sistema são extremamente dinâmicos, com diversos fatores internos e externos ao projeto contribuindo para sua alteração constante. Exemplos de fatores de mudança de requisitos são alterações no ambiente de negócio, uma melhor compreensão do sistema pelos desenvolvedores e a observação do sistema pelos usuários.

Em RUP, a gerência de requisitos consiste em elicitar, organizar e documentar a funcionalidade e as restrições requeridas pelo sistema; avaliar mudanças nesses requisitos e estimar seu impacto; e registrar e documentar decisões.

Com esta prática, RUP pretende atacar riscos de projeto utilizando uma abordagem disciplinada para a gerência de requisitos, comunicando esses requisitos claramente aos envolvidos e oferecendo uma forma de priorização, seleção e rastreamento de requisitos.

2.3.1.3 Usar arquiteturas baseadas em componentes

De acordo com os autores de RUP, a arquitetura é um dos produtos de trabalho mais importantes de um sistema, descrevendo a organização do *software* e a seleção dos elementos estruturais que compõem o sistema; o comportamento, conforme o especificado pela colaboração entre esses elementos; a composição desses elementos estruturais e comportamentais em subsistemas progressivamente maiores.

Uma possível abordagem para a composição da arquitetura de um sistema é o desenvolvimento baseado em componentes. Nesta abordagem, o sistema é organizado em módulos coesos e fracamente acoplados uns aos outros, e então são estudadas possibilidades de aquisição de cada um desses módulos. Cada módulo do sistema pode ser adquirido de um fabricante, desenvolvido pela equipe ou reutilizado de outros projetos (com maior ou menor esforço de adaptação).

Os objetivos do desenvolvimento de uma arquitetura baseada em componentes são: promover a criação de arquiteturas estáveis e robustas, através de utilização de componentes cuja qualidade é comprovada por sua utilização em outros sistemas; dividir o sistema em módulos de forma que sua evolução possa ser feita de forma isolada (alterações em um módulo não precisarão envolver todo o sistema); facilitar o reuso; facilitar a gerência de configuração.

2.3.1.4 Modelar software visualmente

Um modelo é uma simplificação da realidade que descreve um sistema de acordo com uma perspectiva particular. Modelos são usados para compreender melhor um problema ou uma solução, para comunicar uma idéia, para descrever uma abordagem ou alternativas de resolução de questões relacionadas ao sistema; para documentar essas alternativas ou a decisão tomada em prol de uma das alternativas. Todas essas abstrações do sistema permitem que a equipe de desenvolvimento construa o sistema por partes, mantendo o foco na perspectiva que está sendo analisada. Assim, a

modelagem melhora a habilidade da equipe de gerenciar a complexidade do *software*.

Através da visualização dos modelos, comportamentos do sistema podem ser descritos de forma não ambígua, detalhes podem ser abstraídos quando necessário, inconsistências e problemas de arquitetura podem ser encontrados mais facilmente (em oposição à análise de código ou de descrições textuais).

2.3.1.5 Verificar a qualidade do software continuamente

Como o *software* é desenvolvido iterativamente em RUP, ele pode ser verificado constantemente. A verificação se dá por meio de testes em cenários de utilização do sistema. Desta forma, é um processo contínuo de avaliação qualitativa (testes executados com sucesso) e quantitativa (número de testes realizados, número de cenários ainda não testados, número de testes cujo resultado foi bem sucedido, etc.). A verificação contínua de qualidade faz com que o acompanhamento do status do projeto seja mais objetivo, faz com que as inconsistências em requisitos, projeto e implementações sejam evidenciadas, os riscos mitigados, os defeitos encontrados cedo.

2.3.1.6 Controlar as mudanças no software

A principal motivação para esta prática é a crença em que "*manter a rastreabilidade entre os elementos de cada versão liberada e entre os elementos ao longo de liberações múltiplas e paralelas é essencial para avaliar e gerenciar ativamente o impacto de mudanças*". Embora esta crença seja questionada pelos métodos ágeis, descritos mais adiante neste trabalho, é certo que as alterações realizadas em um sistema precisam ser organizadas de alguma forma, utilizando critérios para originar essas mudanças, priorizá-las, escolher seu momento de implantação e avaliar os resultados de cada mudança.

O controle de mudanças em RUP fornece uma seqüência de atividades claramente descritas e padronizadas para execuções repetidas. São definidas as Requisições de Mudança (*Change Requests*), documentos descrevendo criteriosamente as mudanças. A propagação de mudanças é controlada pelo processo e é passível de verificação.

2.3.2 Características de RUP

Além das práticas apresentadas acima, RUP tem ainda três características que norteiam sua utilização:

- É um processo de desenvolvimento orientado a casos de uso;
- Define um *framework* de processo que pode ser adaptado e estendido pela organização que o adota;
- Utiliza largamente o suporte de ferramentas automatizadas.

Casos de uso são cenários de utilização do sistema por usuários. Sua principal meta, no contexto de RUP, é a de caracterizar um fluxo de execução do sistema. O conceito de casos de uso é utilizado por RUP para organizar o desenvolvimento do sistema, sendo refletido em diversas atividades e artefatos.

Conforme explicado anteriormente, o fato de ser um *framework* de processo adaptável faz com que RUP possa ser utilizado de formas diferentes, de acordo

com fatores como a organização que o adota, o tamanho do projeto, a necessidade de integração com outros sistemas internos ou externos à organização, etc. As possibilidades de configuração de RUP são: modificações, customizações, adições ou exclusões em artefatos, atividades, papéis e fluxos de execução de atividades, assim como manuais de orientação e modelos de artefatos (ver Figura 2.3).

O suporte de ferramentas automatizadas a RUP é muito extenso, o que é esperado, já que esta é uma metodologia criada comercialmente, por uma empresa interessada difundir suas ferramentas de suporte ao desenvolvimento de projetos de *software*. O próprio *framework* RUP é documentado como um produto comercial, representado por uma base de conhecimento disponível via navegador web. As ferramentas têm o objetivo de aumentar a produtividade em cada atividade do processo e de facilitar as práticas de RUP.

As práticas e características apresentadas servem como base de fundamentação do *framework* de processo RUP, cuja estrutura é descrita a seguir.

2.3.3 Estrutura de RUP

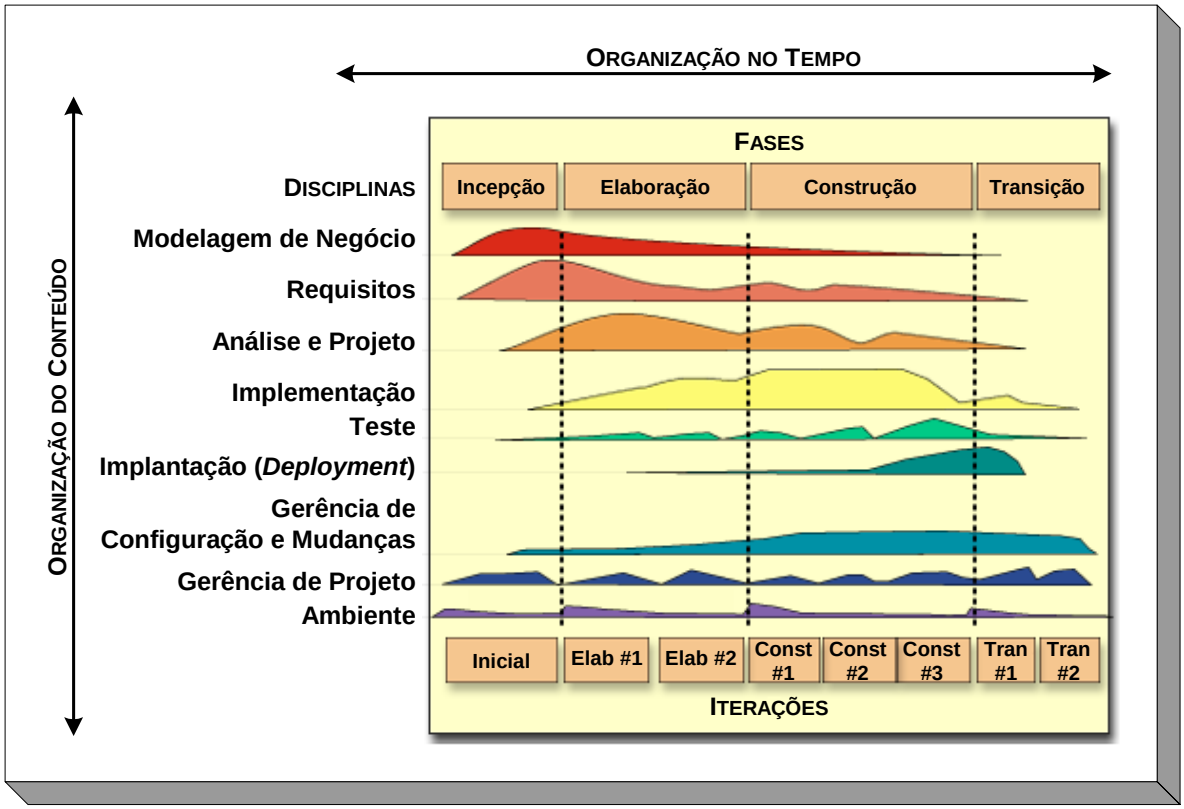


Figura 2.6: Estrutura de Processo de RUP [Adaptada de (KRUCHTEN, 2001)]

A figura acima ilustra a estrutura de RUP. O processo possui duas dimensões: o eixo horizontal, que representa os aspectos dinâmicos do processo, e o eixo vertical, que representa os aspectos estáticos.

A parte dinâmica diz respeito à evolução do projeto ao longo do tempo, é dividida em fases e iterações e planejada de acordo com cada projeto específico. A parte estática descreve as disciplinas do processo, que agrupam atividades logicamente de acordo com sua natureza; é a parte do processo que

descreve *quem* faz *o que*, *como* e *quando* isto é feito. Os gráficos, no centro, representam a carga de trabalho estimada para cada disciplina.

2.3.3.1 Aspectos Estáticos

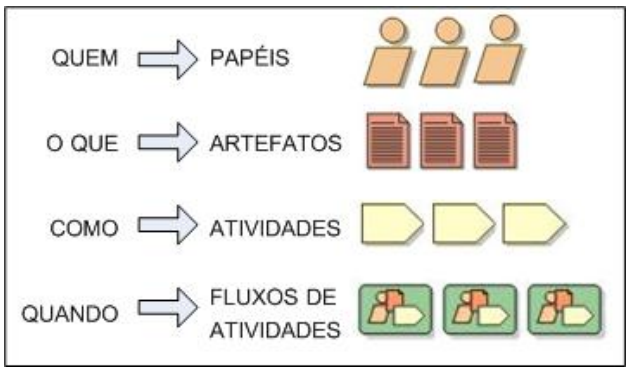


Figura 2.7: Componentes estáticos de RUP

Com relação ao processo estático de RUP, seus componentes são descritos da seguinte forma:

- *Papéis*: a descrição do comportamento e responsabilidades de um indivíduo ou grupo de indivíduos trabalhando juntos em uma equipe de desenvolvimento. O comportamento é definido em termos de atividades que o papel executa, e cada papel é associado a um conjunto de atividades coesas. As responsabilidades de um papel são definidas em relação aos artefatos que devem ser criados, modificados ou controlados. Uma mesma pessoa pode desempenhar vários papéis ao longo do projeto. Exemplos de papéis são o *Analista de Sistemas*, o *Projetista*, o *Projetista de Teste*, etc.
- *Artefatos*: um artefato é uma porção de informação que é produzida, modificada ou utilizada por um processo. Artefatos são os produtos tangíveis de um projeto: as coisas que o projeto produz ou usa enquanto trabalhando rumo ao produto final. Artefatos são usados como entrada por indivíduos desempenhando papéis para executar uma atividade. São usados também como saídas (resultados) de atividades. Podem ser modelos como, por exemplo, diagramas de classe, elementos de modelos como as próprias classes, documentos, código fonte, código executável. Artefatos também podem ser compostos por outros artefatos.
- *Atividades*: definem o trabalho a ser executado por cada pessoa que produz um resultado significativo no contexto do projeto. Cada atividade é associada a um papel específico. A atividade tem um propósito claro, e geralmente incluir criar ou atualizar artefatos. Exemplos de atividades são o *Planejamento de uma Iteração* (*Papel: Gerente de Projeto*), *Encontrar Casos de Uso e Atores* (*Papel: Analista de Sistema*), *Revisar o Projeto* (*Papel: Revisor de Projeto*), etc. Atividades intimamente relacionadas ente si são agrupadas em *Disciplinas* do processo de desenvolvimento – por exemplo, a disciplina de *Requisitos*.
- *Fluxos de Execução de Atividades* (*Workflows*): seqüências de atividades que produzem resultados de valor observável. São representados como diagramas de atividades (um exemplo é

apresentado na Figura 2.8). Cada disciplina de RUP descreve um fluxo de execução de atividades.

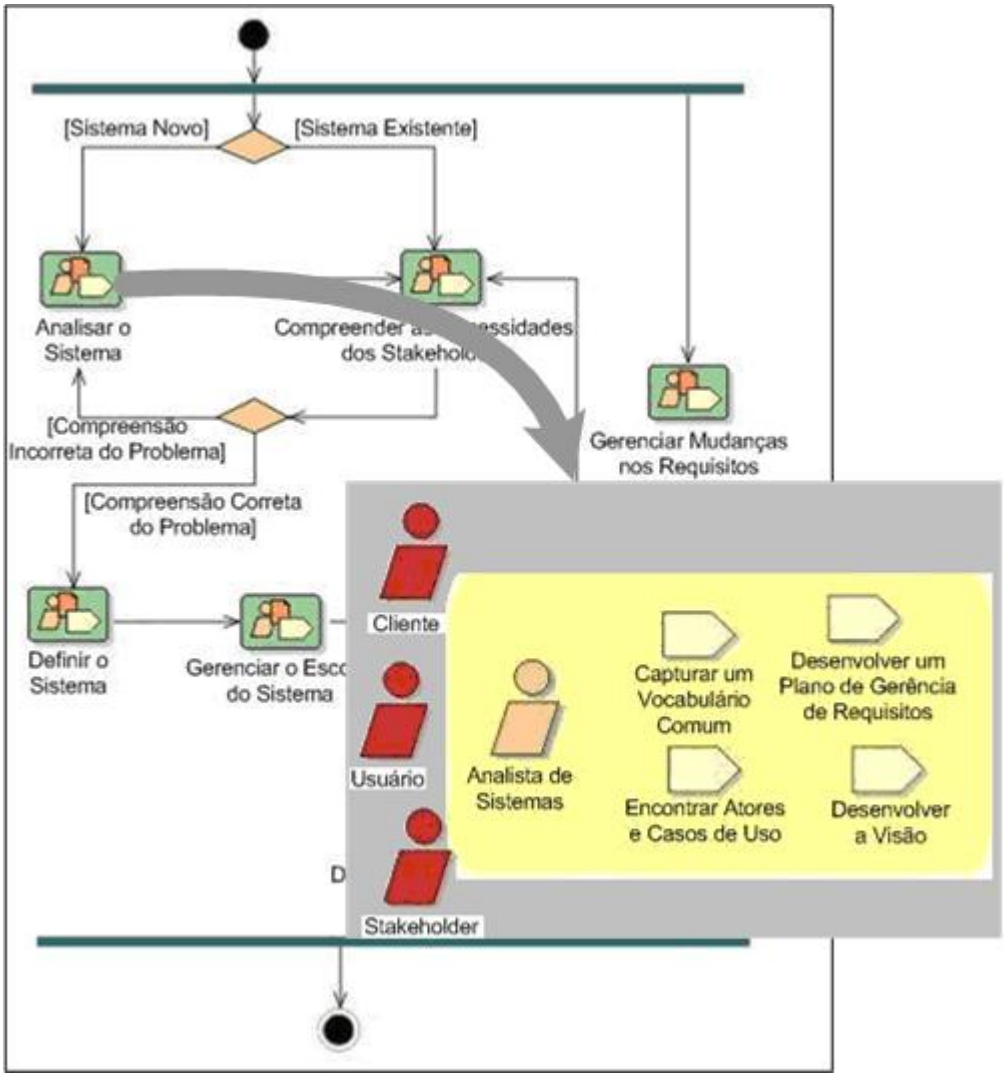


Figura 2.8: Fluxo de Atividades da Disciplina de Requisitos [traduzida de (IBM RATIONAL CORPORATION, 2003)]

2.3.3.2 Aspectos Dinâmicos

Os aspectos dinâmicos de RUP, definidos de acordo com as características e necessidades de cada projeto, são: *fases*, *iterações* e *marcos* (*milestones*).

A idéia por trás das *iterações* é a seguinte: dividir a construção de um grande sistema em um conjunto de sistemas menores, cuja complexidade será menor, o controle e a gerência de riscos facilitados, o *feedback* dos *stakeholders* e desenvolvedores recebido a tempo de gerar alterações no planejamento que garantam o sucesso do projeto. Cada iteração do projeto inclui atividades de análise de requisitos, projeto (*design*), implementação e geração e uma versão do sistema, mesmo que esta versão ainda seja incipiente e incompleta.

Para organizar e orientar as iterações de forma a assegurar convergência para o sistema final resultante, RUP define os *marcos* do ciclo de vida do projeto. Os marcos são pontos ao longo do desenvolvimento do sistema que possuem um objetivo definido que aproxima o sistema em desenvolvimento de

sua finalização. São intimamente relacionados às fases do processo, listadas a seguir:

- *Iniciação ou Concepção* (*Inception*): durante esta fase, é definida a visão geral do produto a ser desenvolvido; a principal preocupação é uma compreensão abrangente dos requisitos do sistema e a determinação do escopo do projeto. A Iniciação termina quando o marco de *Objetivo do Ciclo de Vida* é alcançado: os *stakeholders* concordam com uma definição de escopo para o projeto, e com as estimativas de custo e prazo; os requisitos estão compreendidos e documentados; existe credibilidade quanto às estimativas de custo e prazo, e quanto a prioridades, riscos e o processo de desenvolvimento.
- *Elaboração* (*Elaboration*): o foco desta fase engloba três aspectos: o planejamento das atividades e recursos necessários ao projeto, a especificação detalhada dos requisitos e o projeto da arquitetura do sistema. As atividades de geração de código desta fase são voltadas à criação de um protótipo da arquitetura, à mitigação de riscos técnicos através da experimentação de possíveis soluções, e ao treinamento em ferramentas e técnicas específicas. O marco que sinaliza a finalização da Elaboração é a *Arquitetura do Ciclo de Vida*, que garante a estabilidade da visão e da arquitetura do produto, e a concordância dos envolvidos quanto aos planos para a entrega do sistema.
- *Construção* (*Construction*): é responsável pela evolução do sistema e de sua arquitetura, até que o sistema esteja pronto para entrega à comunidade de usuários. As principais preocupações são o projeto (*design*) e a implementação do sistema. O marco final da Construção é a *Capacidade Operacional Inicial*, atingido quando há uma versão estável e madura para entrega e quando os *stakeholders* estão prontos para a execução desta entrega.
- *Transição* (*Transition*): nesta fase, o sistema é repassado a seus usuários. São resolvidas as questões de empacotamento, entrega, treinamento, suporte e manutenção. A conclusão da Transição se dá quando o marco da *Liberação do Produto* (*Product Release*) é alcançado, o que significa que o cliente está satisfeito com o sistema entregue.

Dentro de cada fase, são definidas uma ou mais iterações. O foco das atividades executadas dentro de cada iteração varia de acordo com os objetivos da fase.

2.3.4 Customização de RUP

O RUP define 4 fases, 9 disciplinas, mais de 40 papéis e mais de 100 artefatos; o produto comercial RUP possui mais de mil páginas de orientação. Além disso, ainda é possível encontrar processos para adicionar funcionalidades e conteúdo à versão padrão. Com toda esta bagagem, RUP é considerado por muitos como muito *pesado* (*heavyweight*) (TAFT, 2005), (KRUCHTEN, 2001), (EVANS, 2003), já que ele prescreve muitas atividades e artefatos a ser desenvolvidos ao longo do processo. Porém, RUP disponibiliza um mecanismo de configuração de processo cujo objetivo é justamente definir o subconjunto de artefatos, papéis e atividades que realmente são necessários.

Na própria definição de RUP (IBM RATIONAL CORPORATION, 2003), menciona-se que nenhum projeto pode, por si só, beneficiar-se da utilização do *framework* RUP *completo*. A utilização de todos os papéis, artefatos, atividades e orientações de RUP padrão em um único projeto provavelmente resultaria em um ambiente de desenvolvimento ineficiente, onde seria difícil manter o foco nas atividades e artefatos mais importantes.

RUP pode ser modificado de forma a melhor atender às necessidades do projeto ou da organização que o adota. Esta adaptação pode ser feita em dois níveis:

- Nível de *organização*: considerando questões como domínio de aplicações, práticas de reuso e tecnologias dominadas pela organização, RUP é modificado e configurado levando em conta processos utilizados por toda a organização.
- Nível de *projeto*: mantendo o foco em um projeto específico, seu tamanho, possibilidades de reuso a partir de repositórios da organização, o fato de ser um projeto novo ou uma nova versão de um sistema já construído, a instância de RUP adotada pela organização é refinada para o projeto.

De acordo com as orientações descritas em (KRUCHTEN, 2001), RUP pode sofrer três tipos de adaptação: *extensão*, *configuração* ou *instanciação*. A extensão e a configuração envolvem uma modificação do processo padrão.

Estender RUP significa adicionar pontos ao processo que enderecem necessidades específicas do projeto ou da organização que não são tratadas a contento por RUP padrão. *Configurar* RUP envolve tomar decisões como a seleção de componentes relevantes do processo, a eliminação de elementos desnecessários, a definição de recursos para a geração de artefatos pela organização e a definição de visões diferentes do processo para cada equipe envolvida no processo.

Instanciar RUP significa definir um processo, a partir do *framework* RUP, a ser utilizado em determinada organização ou em determinado projeto, definindo os artefatos que serão produzidos, coletando e customizando orientações, definindo o modelo de ciclo de vida a ser utilizado, etc. (ver Figura 2.3).

Neste trabalho, o termo *instância de RUP* será utilizado para se referir a um modelo de RUP estendido, configurado e instanciado conforme as necessidades da organização.

2.4 Métodos Ágeis

Até agora, foram apresentados dois enfoques utilizados como fundamentação teórica na definição do processo de engenharia de requisitos proposto neste trabalho: CMMI, um modelo para melhoria de processos descrito através de metas e práticas de desenvolvimento, e RUP, um *framework* de processo de desenvolvimento que contém orientações sobre aspectos dinâmicos (iterações, fases e marcos) e aspectos estáticos (atividades, artefatos, papéis e fluxos de atividades) de um projeto. O terceiro enfoque estudado neste trabalho são os métodos ágeis, uma série enfoques de desenvolvimento de software que possuem, como uma de suas principais características, a possibilidade de flexibilizar a delimitação de escopo de projetos de *software*, abrindo mão da definição e documentação do conjunto

completo de requisitos do sistema em uma etapa inicial do projeto (WILLIAMS; COCKBURN, 2003).

A motivação para o desenvolvimento dos métodos ágeis é a realidade de negócios do século 21, caracterizada pela globalização, por um grande número de fusões e aquisições, e por um ambiente com mudanças freqüentes em objetivos, mercados e estrutura das organizações, além do panorama tecnológico, em constante e acelerada evolução (LYCETT et al., 2003). O cenário descrito evidencia problemas nos processos de desenvolvimento de *software* cuja premissa é a de que os requisitos do sistema podem ser definidos em uma etapa prévia à sua construção. Este tipo de processo de desenvolvimento tende a trabalhar com requisitos e planos desatualizados, mesmo em pequenos espaços de tempo (WILLIAMS; COCKBURN, 2003). Há, ainda, a questão de requisitos desconhecidos por todos os interessados no projeto pela ignorância das implicações das novas tecnologias e da aplicação sendo construída sobre estas. Exemplos deste cenário são aplicações de voz sobre IP, gerência de redes convergentes de dados, voz e imagens, aplicações convergentes de telefones móveis, integração de sistemas de *Enterprise Resource Planning* (ERP) a sistemas legados de grandes corporações, etc.

Exemplos desses métodos, criados a partir da metade da década de 1990, são (WILLIAMS; COCKBURN, 2003): Método Dinâmico de Desenvolvimento de Sistemas (DSDM – *Dynamic Systems Development Method*) (FAZACKERLEY, 2005), Desenvolvimento Guiado por Funcionalidade (FDD – *Feature-Driven Development*) (PALMER; FELSING, 2002), *Extreme Programming* (XP) (BECK, 2000), Crystal (COCKBURN, 2004), Desenvolvimento de Software Adaptativo (*Adaptive Software Development*) (HIGHSMITH, 1999) e Scrum (SCHWABER, 2004),.

Os criadores dos métodos ágeis debateram os princípios que utilizaram na definição de seus processos e perceberam que todos partiram de uma base comum de idéias. Então, formalizaram essas idéias no *Manifesto pelo Desenvolvimento Ágil de Software*.

2.4.1 O Manifesto pelo Desenvolvimento Ágil de Software

O *Manifesto pelo Desenvolvimento Ágil de Software* (*Manifesto for Agile Software Development*), ou simplesmente *Manifesto Ágil*, formaliza os princípios básicos que dão suporte aos métodos ágeis de desenvolvimento de *software*. O termo ágil (*agile*) foi adotado pelos criadores desses métodos, que formaram a *Aliança Ágil* (*Agile Alliance*), e disponibilizaram suas idéias, bem como diversos recursos relacionados a elas, em um *web site*: (ANDREA et al., 2005). O Manifesto Ágil resume-se às linhas do quadro apresentado abaixo (BECK et al., 2001).

MANIFESTO PELO DESENVOLVIMENTO ÁGIL DE SOFTWARE

Nós estamos descobrindo melhores formas de desenvolver *software* enquanto o fazemos e enquanto ajudamos os outros a fazê-lo. Através deste trabalho nós viemos a valorizar:

Indivíduos e interações sobre processos e ferramentas

***Software* funcionando** sobre documentação abrangente

Colaboração do cliente sobre negociação contratual

Responder a mudanças sobre seguir um plano

Ou seja, embora haja valor nos itens à direita, nós damos mais valor aos itens à esquerda.

Figura 2.9: O Manifesto pelo Desenvolvimento Ágil de Software (BECK et al., 2001)

Os princípios fundamentais que dão suporte aos métodos ágeis, também publicados em (BECK et al., 2001), são os seguintes:

- *A maior prioridade é satisfazer o cliente através da entrega pronta e contínua de software com valor agregado.*
- *Receba bem as alterações em requisitos, mesmo tarde no desenvolvimento. Processos ágeis suportam mudanças para a vantagem competitiva do cliente.*
- *Entregue software funcionando freqüentemente, de algumas semanas a alguns poucos meses, com preferência para a escala de tempo mais curta.*
- *Interessados na aplicação (stakeholders) e desenvolvedores devem trabalhar juntos diariamente ao longo do projeto.*
- *Construa projetos ao redor de indivíduos motivados. Dê a eles o ambiente e o suporte que eles precisam, e confie neles para que façam o serviço.*
- *O método mais eficiente e eficaz de conduzir informações para e dentro de um time de desenvolvimento é a comunicação face-a-face.*
- *Software funcionando é a medida principal de progresso.*
- *Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores, e usuários devem ser capazes de manter um ritmo constante indefinidamente.*
- *Atenção contínua à excelência técnica e ao bom projeto aumenta a agilidade.*
- *Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.*
- *As melhores arquiteturas, requisitos, e projetos surgem de times auto-organizados.*
- *Em intervalos regulares, o time reflete sobre como pode ser tornar mais eficaz, então melhora e ajusta seu comportamento de acordo com isso.*

Os valores e princípios do Manifesto Ágil são a fundamentação dos métodos ágeis. Nas próximas seções, serão brevemente descritos dois desses métodos: *Extreme Programming* (XP) e *Modelagem Ágil* (*AM – Agile Modeling*). O objetivo da descrição desses métodos é exemplificar a utilização das práticas e princípios ágeis em processos de desenvolvimento de *software*. A Modelagem Ágil, especificamente, discute diversas práticas que podem ser empregadas no processo de engenharia de requisitos proposto neste trabalho.

2.4.2 EXtreme Programming (XP)

*E**X**treme **P**rogramming* (XP), ou *Programação Extrema* (BECK, 2000), é um método ágil para desenvolvimento de *software*, caracterizado por: iterações curtas, com entrega de *software* aos usuários, cuja avaliação do sistema e necessidades de negócio são usadas como entrada para o planejamento da iteração seguinte; flexibilidade e habilidade para acomodar mudanças de negócio; utilização de testes automatizados para monitorar o progresso do sistema; foco na comunicação entre todos os envolvidos no projeto; constante evolução da arquitetura, ao longo do desenvolvimento do sistema.

Assim como outras iniciativas de desenvolvimento de *software* ágil, XP baseia-se em um conjunto sucinto de valores, princípios e práticas fundamentais que orientam as atividades realizadas durante o desenvolvimento. Scott Ambler descreve esta abordagem da seguinte forma (AMBLER, 2002): "*Inicia-se com um conjunto de valores de alto nível, define-se uma coleção de princípios concretos baseados nesses valores, e então formulam-se práticas para esses valores e princípios que devem ser aplicadas no trabalho*".

Uma das razões pelas quais XP tem despertado grande atenção ultimamente é a associação feita entre seu nome, *Programação Extrema*, e o desenvolvimento de *software* sem planejamento e sem preocupações com projeto (*design*) de *software* (LARMAN; BASILI, 2003). Na verdade, esta é uma associação errônea. A palavra "*Extrema*" se refere à utilização dos princípios e práticas de XP em níveis extremos e não à atividade específica da programação. Os valores, princípios e práticas de XP são descritos a seguir, seguidos de breve descrição das atividades que integram este método. A principal fonte de informação utilizada nestas seções é o livro *Extreme Programming Explained*, do criador de XP, Kent Beck (BECK, 2000).

2.4.2.1 Valores de XP

Os valores são conceitos básicos que orientam o método a ser seguido; resumem a filosofia de trabalho e a orientação a ser utilizada na tomada de decisões. XP possui quatro valores:

- *Comunicação*: o andamento do projeto é comunicado a todos os interessados constantemente. Os primeiros sinais de problemas são repassados a todos, assim como as dúvidas e preocupações. O conhecimento do sistema cresce conjuntamente, com a interação de todos os integrantes da equipe (desenvolvedores e clientes).
- *Simplicidade*: há uma preocupação extrema com a procura da "*coisa mais simples que possa funcionar*". Especificações e código complexos são desencorajados, na busca por soluções de fácil entendimento, implementação e manutenção.

- *Retroalimentação* (*Feedback*): todas as teorias são testadas tão cedo quanto possível. Quando não se pode mais compreender uma especificação sem implementá-la, parte-se para o código. Quando a implementação uma funcionalidade requisitada pelo usuário é terminada, o usuário deve validá-la. Estas estratégias são utilizadas para garantir que o projeto caminhe sempre para o ponto esperado por todos.
- *Coragem*: práticas ágeis de desenvolvimento de *software* são inovadoras e requerem alta confiança no processo para ser empregadas corretamente. O valor da *coragem* representa o compromisso necessário com o método para aplicá-lo nos diversos cenários do cotidiano.

Estes valores básicos são utilizados para definir os princípios de conduta a ser adotados pelo método de XP.

2.4.2.2 Princípios de XP

Com base nos valores, são definidos princípios concretos de XP que ajudam no momento de tomar decisões entre abordagens alternativas. Se existem duas soluções possíveis para um problema, escolhe-se a que melhor preserva os princípios envolvidos. Os princípios básicos de XP são:

- *Retroalimentação rápida* (*rapid feedback*): quanto mais cedo se obtém o retorno sobre algo que foi feito, melhor se avalia se o resultado obtido foi satisfatório. Assim, quanto mais rápida for a retroalimentação, mais efetivo será o aprendizado.
- *Assumir simplicidade*: deve-se procurar pela solução mais simples possível para cada problema. Projetos simples são mais fáceis de implementar e podem ser alterados em caso de necessidade.
- *Mudanças incrementais*: pequenas mudanças podem ser implementadas rapidamente e com baixo risco. Problemas advindos de mudanças pequenas são encontrados com facilidade quando a alteração é limitada. Mudanças, em XP, são feitas aos poucos, em uma série de pequenos esforços.
- *Mudanças são bem-vindas*: durante o desenvolvimento de *software*, diversas mudanças ocorrem em vários âmbitos. A forma mais produtiva de trabalhar neste ambiente dinâmico é encarar as mudanças como bem-vindas e incorporá-las ao processo de desenvolvimento.
- *Trabalho de qualidade*: este princípio tem efeito em praticamente todas as atividades realizadas em um projeto de *software*. A qualidade do código produzido facilita seu entendimento, teste e manutenção. O trabalho de qualidade contribui, ainda, com o bem estar da equipe, melhorando sua produtividade e sua satisfação com o processo.

Além dos princípios listados acima, existem princípios complementares de XP, que ajudam na orientação a ser seguida durante as atividades do processo de desenvolvimento:

- *Ensinar a aprender*: ao invés de definir as ações a ser tomadas em cada situação do projeto, XP promove o ensino de estratégias de decisão, a partir dos valores, princípios e práticas do método.
- *Investimento inicial pequeno*: iniciando-se com um pequeno investimento (ex: poucos membros na equipe, apenas as ferramentes e máquinas

necessárias para a primeira liberação), pode-se manter o foco do projeto mais definido, sem maiores necessidades de gerenciamento.

- *Jogar para vencer*: quando se está confiante no processo de desenvolvimento, todas as atividades são feitas da forma correta, sem a queima de etapas como testes ou preocupação com qualidade para cumprir as metas estipuladas.
- *Experimentos concretos*: a experimentação, representada principalmente pela implementação e pelo teste, diminui consideravelmente o risco de calcar o projeto em premissas falsas.
- *Comunicação aberta e honesta*: a comunicação é sempre incentivada: seja para informar novas funcionalidades necessárias, seja para relatar que um prazo não será cumprido, seja para expressar suspeitas de que certa porção de código foi escrita de forma que resultará em problemas futuros.
- *Trabalhar com os instintos das pessoas, não contra eles*: os interesses de longo prazo do projeto são traduzidos em interesses de curto prazo para os desenvolvedores. A comunicação de requisitos, por exemplo, é feita através do planejamento, dos testes de aceitação, da conversação informal com o cliente, que são recursos utilizados instintivamente, e não através de extensa documentação.
- *Responsabilidade aceita*: as tarefas a ser executadas são identificadas e listadas, e cada membro da equipe requisita suas próprias tarefas, ficando responsável por sua estimativa de esforço.
- *Adaptação local*: cada organização tem sua própria cultura e um processo de desenvolvimento que se insira nessa cultura local deve ser criado para seus projetos.
- *Carregar pouca bagagem (travel light)*: os artefatos do projeto devem ser poucos, simples e importantes. Alguns documentos podem ser criados apenas para compreender ou comunicar um problema ou solução, e podem ser descartados após cumprirem seu propósito.
- *Medidas honestas*: estimativas são baseadas em experiências prévias e em explorações feitas com protótipos. O cronograma pode não ser exato, mas deve conter valores com embasamento concreto e não palpites ou prazos resultantes de determinações externas ao projeto.

Com esses princípios em mente, definem-se as ações a ser postas em prática durante o desenvolvimento.

2.4.2.3 Práticas de XP

Com base nos princípios descritos anteriormente, são criadas práticas, hábitos que devem ser exercitados durante o desenvolvimento de *software* em XP:

- *O Jogo do Planejamento (The Planning Game)*: é a principal forma de gerência de XP, e define regras sobre como fazer o planejamento do projeto. As regras permitem dois tipos de decisão: a equipe técnica determina a complexidade das funcionalidades requisitadas para o sistema e estima seu custo (em termos de tempo de desenvolvimento); já a equipe de negócio, ou seja, o cliente, decide quais são as

necessidades do negócio em termos de escopo, prioridade e datas de entrega de versões.

- *Pequenas Liberações*: cada liberação de versão tem o menor escopo possível que resolva um conjunto relevante de requisitos de negócio. A primeira versão do sistema deve conter apenas suas principais funcionalidades. Após essa, as próximas versões agregam valor ao sistema em ciclos tão curtos quanto possível.
- *Metáfora*: todo o desenvolvimento é guiado por uma metáfora, uma história que explica como o sistema funciona. O objetivo da metáfora é servir como orientação para decisões arquiteturais sem a necessidade de uma documentação extensa e custosa destas decisões.
- *Projeto Simples*: uma inovação de XP com relação a outras abordagens de desenvolvimento de *software* é uma busca pelo projeto mais simples possível. Para que o sistema evolua quando necessário, é posto em prática a *refatoração* (descrita a seguir).
- *Teste*: para assegurar a correção do sistema, programadores escrevem testes unitários continuamente. Um desenvolvedor só implementa a próxima funcionalidade quando todos os testes executarem sem erros para o que acabou de ser desenvolvido. Clientes escrevem testes funcionais para confirmar que requisitos do sistema foram satisfeitos.
- *Refatoração (Refactoring)*: uma das práticas de XP é o *projeto simples*. Esta prática mantém o foco das atividades de projeto (*design*) nas funcionalidades de cada versão do sistema, sem prever possíveis necessidades futuras. Quando as funcionalidades de uma determinada versão evidenciam que o projeto deveria ter sido mais elaborado, este projeto deve evoluir. A evolução controlada do projeto do sistema, adicionando flexibilidade sem alterar o comportamento corrente, é chamada de *refatoração*.
- *Programação em Pares*: o código é produzido por pares de programadores, cada par utilizando uma mesma máquina. Cada par tem dois papéis: um programador escreve o código e o outro observa a linha de programação adotada, avaliando o projeto escolhido, imaginando que outros testes poderiam ser construídos, etc. Os programadores alternam os papéis constantemente. Os pares também são alterados de forma bastante dinâmica.
- *Propriedade Coletiva do Código Fonte*: "*qualquer um pode alterar qualquer parte do código a qualquer momento*". Esta prática se apóia firmemente em outras como os testes automáticos, a integração contínua e a utilização de padrões de codificação, que tornam seguro o fato de um programador alterar código que não foi criado por ele.
- *Integração Contínua*: o sistema é integrado várias vezes ao dia, após a conclusão de cada tarefa. Após integrar seu código recém terminado ao restante do sistema, o desenvolvedor executa os testes automáticos. Se algum problema for sinalizado, o programador o corrige antes passar à sua próxima tarefa.
- *Semana de 40 horas*: estar bem disposto influi diretamente na capacidade de trabalho, produtividade e criatividade de cada um. XP aconselha que se procure não extrapolar o limite de carga de trabalho,

embora admita que uma quebra desta regra pode ser necessária uma vez ou outra (nunca por mais de uma semana seguida).

- *Cliente Presente*: um cliente do sistema faz parte da equipe e pode ser consultado a qualquer momento. Este cliente utilizará o sistema quando este estiver pronto e deve estar preparado para tomar decisões sobre funcionalidades, para escolher entre possíveis estratégias de apresentação das funcionalidades, ou para priorizar necessidades. Esta prática procura reduzir um dos riscos clássicos de projetos: o de não satisfazer as necessidades de seus usuários.
- *Padrões de Codificação*: além de ser uma boa prática de programação, a utilização de padrões fornece uma identidade comum ao código, favorecendo práticas como a posse coletiva, a comunicação através do código e a refatoração.

As práticas complementam e promovem umas às outras, compensando desvantagens que cada prática isolada pode trazer.

2.4.2.4 Atividades de XP

Juntamente com as práticas, as atividades definem como se dará o desenvolvimento do *software*. Diferente de metodologias tradicionais, XP não define um conjunto amplo de atividades, papéis, artefatos, pontos de sincronização, entradas e saídas. Mantendo o valor da simplicidade, apenas estas quatro atividades são definidas, sem orientações sobre seu encadeamento durante o processo de desenvolvimento:

- *Codificar*: além de ser o objetivo principal em um projeto de desenvolvimento de *software*, o código é também uma excelente forma de entender, aprender, comunicar e testar. A possível solução de um problema pode ser compreendida implementando-se algumas versões de código dessa solução. Uma maneira de explicar uma idéia aos outros é mostrar como o código se pareceria. Para atestar a correção de uma funcionalidade, pode-se implementar testes automáticos que a comprovem.
- *Testar*: os testes são criados principalmente para comprovar que o código construído está correto. Porém, há muitos outros benefícios que podem ser obtidos em função da construção dos testes: construir os testes antes da implementação dá ao desenvolvedor uma idéia geral do que deve ser construído; um sistema que possui testes automáticos pode ser alterado sem que isso gere insegurança, já que basta comprovar que os testes continuam indicando uma execução correta para saber-se que as alterações foram bem sucedidas; após algum tempo cultivando o hábito de criação extensiva de testes, a produtividade de um desenvolvedor acaba aumentando, já que o tempo de correção (e identificação) de defeitos diminui.
- *Ouvir*: para que se tenha certeza de que o *software* produzido atende às necessidades, é preciso ouvir o que seus usuários têm a dizer. Sendo assim, esta atividade encoraja a compreensão das necessidades do cliente, bem como o retorno dado por ele ao avaliar cada funcionalidade desenvolvida no sistema.
- *Projetar (Designing)*: esta atividade tem o objetivo de organizar a lógica do sistema de forma que este se torne estável e flexível. Utilizar boas

práticas de projeto é a única forma de garantir que o *software* poderá evoluir sem causar sérios efeitos colaterais.

2.4.2.5 Resumo do Método XP

A figura a seguir resume os valores, princípios, práticas e atividades de XP. Uma definição detalhada deste método pode ser encontrada em (BECK, 2000).

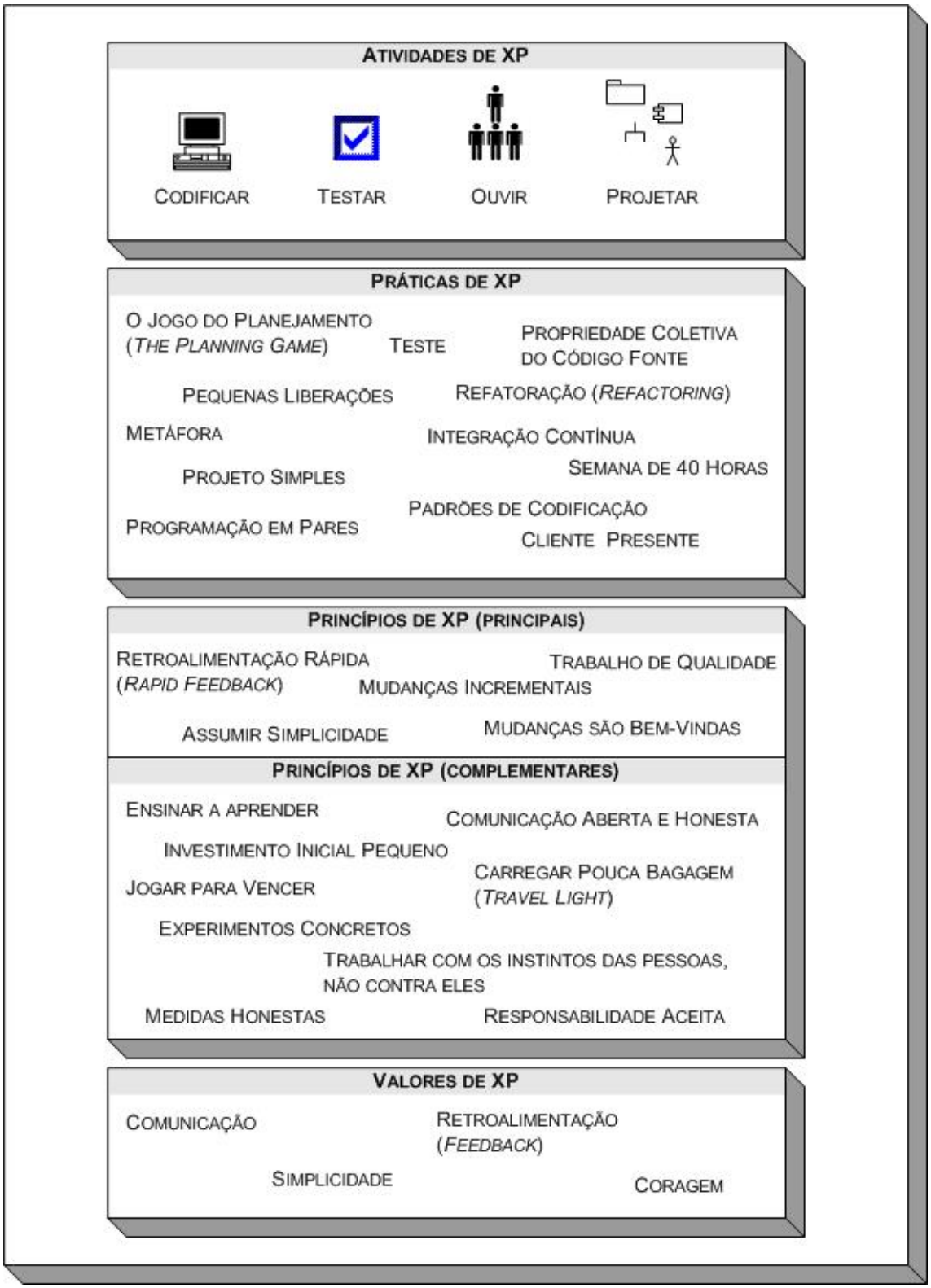


Figura 2.10: Resumo do Método XP

2.4.3 Modelagem Ágil

A Modelagem Ágil é uma abordagem de desenvolvimento baseada em práticas para a modelagem e documentação efetiva de sistemas de *software* calcada nos princípios promovidos pela Aliança Ágil.

Segundo (AMBLER, 2002), modelos são utilizados para dois fins: para entender e desenvolver conhecimento mais amplo sobre determinado problema ou solução e para comunicar este entendimento a outros. A proposta da Modelagem Ágil é definir práticas e orientações para modelar de forma eficaz e eficiente, com os seguintes objetivos:

1. Definir e orientar a utilização de valores, princípios e práticas para modelagem eficaz e *leve* (*lightweight*). A inovação não estaria nas técnicas de modelagem, já que estas já são utilizadas por diversos outros métodos, mas sim na forma de aplicá-las.
2. Tratar o problema da aplicação de técnicas de modelagem em projetos de *software* utilizando uma abordagem ágil, seguindo as orientações de *Aliança Ágil*.
3. Sugerir como processos que são instâncias de RUP podem ser configurados de forma a adotar princípios ágeis no que diz respeito a atividades de modelagem.

A forma de definição da Modelagem Ágil é inspirada em XP, e muitos conceitos, valores, princípios e práticas são simplesmente iguais aos definidos em XP (ver seção sobre *Extreme Programming*). As próximas seções descrevem alguns detalhes da modelagem ágil, seus valores, princípios e práticas e suas formas de aplicação. Estas informações foram obtidas de (AMBLER, 2002).

2.4.3.1 Valores da Modelagem Ágil

Os valores que dão suporte à Modelagem Ágil são: comunicação, simplicidade, retroalimentação (*feedback*), coragem e humildade. Comparando-os aos valores de XP, percebe-se que a inovação nesta abordagem é a inclusão do valor da *humildade*. O objetivo deste valor é lembrar ao desenvolvedor que todos podem cometer enganos, que é possível que uma abordagem sugerida por outra pessoa seja melhor para resolver determinado problema, que é possível aprender com todos os outros envolvidos no processo (outros desenvolvedores e *stakeholders*, por exemplo). Outro efeito da humildade como valor é o fato de respeitar outros pontos de vista sobre o sistema, já que cada envolvido terá suas próprias experiências e prioridades. O cultivo da receptividade para com outros melhora a cooperação e a comunicação no âmbito geral do projeto.

Da mesma forma que com XP, estes valores são conceitos abstratos que servem como fundamentação para os princípios da abordagem Modelagem Ágil, apresentados a seguir.

2.4.3.2 Princípios da Modelagem Ágil

Alguns princípios da Modelagem Ágil são adotados entre os princípios de XP, e estão descritos na sessão de Princípios de XP: carregar pouca bagagem (*travel light*), assumir simplicidade, mudanças são bem-vindas, mudanças incrementais, efetuar um trabalho de qualidade.

Além destes, a abordagem da Modelagem Ágil utiliza, ainda, os seguintes princípios:

- *O software é o objetivo principal*: este princípio é inspirado no valor do Manifesto Ágil "*software funcionando sobre documentação abrangente*" e no princípio derivado deste valor "*software funcionando é a medida principal de progresso*". O principal artefato a ser entregue ao cliente é o *software*. O investimento em modelos e documentação é válido em diversas situações, mas o sistema em funcionamento é indispensável.
- *Habilitar o próximo esforço é o objetivo secundário*: o próximo esforço pode ser desenvolver a próxima versão do sistema ou apenas operar e fornecer manutenção à última versão construída. Para isso, não basta apenas desenvolver *software* de qualidade. É preciso disponibilizar o mínimo de documentação necessária ao pessoal que realizará manutenção, ou à própria equipe que dará seguimento à construção da nova versão do sistema; é preciso transferir o conhecimento entre equipes do projeto ou até mesmo motivar a equipe atual a continuar suas atividades.
- *Modelar com um objetivo*: quando um modelo está sendo criado, deve estar claro qual será sua audiência e qual seu motivo de criação. A audiência pode ser outros desenvolvedores, gerentes, membros de outros projetos que serão integrados ao *software* descrito pelo modelo, a equipe de manutenção, etc. O motivo pode ser explicar uma abordagem a um colega, compreender melhor uma solução, documentar o sistema para o próximo esforço. A audiência e o motivo de criação de um documento são usados para definir o nível de detalhamento, a tolerância de inconsistências e outros fatores como estes.

  Outro aspecto a ser levado em conta na criação de modelos é o fato de que, quando estes atingem seus objetivos, deve-se encerrar o trabalho no modelo. Este é provavelmente o ponto ótimo de custo/benefício que este modelo pode atingir.
- *Utilizar múltiplos modelos em paralelo*: diversas técnicas de modelagem de sistemas disponibilizam um amplo número de modelos para serem usados por desenvolvedores. Existem os modelos da UML (*Unified Modeling Language*) como diagramas de casos de uso, diagramas de classe e de seqüência, modelos de dados, protótipos de interface de usuário, modelos navegacionais, e muitos outros. A Modelagem Ágil sugere que vários destes modelos sejam construídos e utilizados em paralelo. Por exemplo, ao criar diagramas de casos de uso, é possível definir as classes (com seus métodos e atributos) que darão suporte a determinado caso de uso, além de diagramas de seqüência que representam os diversos cenários possíveis para o caso de uso. Atualizações em um modelo evidenciam as necessidades dos demais.
- *Maximizar o investimento dos stakeholders*: um dos princípios básicos da modelagem ágil é a maximização do investimento dos *stakeholders*. Os *stakeholders* do projeto estão investindo recursos – tempo, dinheiro, logística, etc. – para que seja desenvolvido um sistema que supra suas necessidades. Logo, têm o direito de investir seus recursos da melhor forma possível. Isso significa que os *stakeholders* devem tomar a decisão final sobre investir ou não seus recursos em uma determinada

atividade ou artefato. Aplicado à modelagem e à documentação, o poder de decisão dos *stakeholders* significa que eles (e não os desenvolvedores) é que devem decidir que modelos e documentos serão mantidos, quais serão atualizados, que ferramentas serão utilizadas, e assim por diante.

Com base nestes princípios, são definidas as práticas, que serão executadas ao longo das atividades do desenvolvimento do sistema.

2.4.3.3 Práticas da Modelagem Ágil

As práticas da modelagem ágil são divididas em quatro categorias:

- *Modelagem Iterativa e Incremental*: aplicar os artefatos corretos, criar vários modelos em paralelo, iterar para outro artefato, modelar em pequenos incrementos.
- *Trabalho em Equipe* (*Teamwork*): modelar com os demais, adotar participação ativa dos *stakeholders*, praticar a posse coletiva, exibir os modelos publicamente.
- *Simplicidade*: criar conteúdo simples, descrever os modelos de forma simples, usar as ferramentas mais simples.
- *Validação*: considerar a habilidade de efetuar testes, provar com código.

A maioria destas práticas não representa uma novidade, mas sim um conjunto de melhores práticas seguido por desenvolvedores ao longo dos anos em seus projetos. A contribuição da Modelagem Ágil é reunir estas práticas de forma estruturada e orientar sua utilização de forma que seja formada uma sinergia em prol da modelagem eficaz de sistemas de *software*. A seguir, cada prática é comentada em separado:

- *Aplicar os artefatos corretos*: cada modelo tem uma área específica de atuação, uma situação ideal de utilização. Esta prática orienta que os artefatos sejam escolhidos de acordo com o problema a ser modelado. Não se deve usar sempre o mesmo conjunto de modelos, ditados por algum método de desenvolvimento, mas sim os que se adequarem ao sistema e à solução que está sendo criada.
- *Criar vários modelos em paralelo*: inspirada no princípio *utilizar múltiplos modelos em paralelo*, esta prática é utilizada juntamente com *itere para outro artefato*. O principal objetivo é utilizar o que cada modelo tem a contribuir para a solução de um problema, construindo vários pontos de vista sobre a mesma solução. Não deve ser esquecido, porém, que não é necessário desenvolver-se todos os modelos possíveis em um único projeto. Cada modelo utilizado deve ter um motivo relevante para ser criado.
- *Iterar para outro artefato*: durante a construção de um artefato de modelagem, este é desenvolvido até que não se possa pensar em mais nenhuma informação a ser adicionada. Porém, quando se passa a desenvolver outro modelo, é comum encontrar pequenos problemas ou notar a ausência de algumas informações no primeiro artefato. Assim, se dois (ou mais) modelos são desenvolvidos iterativamente, um complementa o outro, tornando a modelagem mais completa e eficiente.
- *Modelar em pequenos incrementos*: a idéia básica é dividir um amplo esforço em pequenas porções que são entregues ao longo do tempo,

idealmente em incrementos de poucas semanas ou poucos meses. Isso faz com que o *software* seja entregue aos usuários mais rapidamente, e permite que a retroalimentação (*feedback*) também seja recebida logo. A próxima iteração se alimenta dos resultados da iteração atual, e do conhecimento do sistema promovido por ela.

- *Modelar com os outros*: o principal objetivo desta prática é promover a retroalimentação (*feedback*) imediata, a troca de idéias e experiências entre desenvolvedores e, principalmente, a comunicação. Além disso, esta prática favorece a criação de um vocabulário comum entre os envolvidos no projeto.
- *Adotar participação ativa dos stakeholders*: a participação ativa dos *stakeholders* se traduz no compartilhamento de informações sobre o negócio; na tomada de decisões relevantes sobre o escopo do sistema e sobre as prioridades dos requisitos de forma rápida e condizente com as necessidades do projeto; em realmente fazer parte da equipe de desenvolvimento, conhecendo as possibilidades, tecnologias, dedicação e qualificação da equipe.
- *Praticar a posse coletiva (collective ownership)*: qualquer um pode alterar qualquer artefato a qualquer momento, desde que esteja certo de que sua contribuição fará com que o artefato ilustre melhor suas informações. Esta prática estimula a retroalimentação, a utilização de melhores práticas, a comunicação e o aprendizado coletivo da equipe.
- *Exibir os modelos publicamente*: este é mais um passo a favor da comunicação e do estímulo à retroalimentação (*feedback*). O local de exibição pode variar de uma parede a um *web site* com os modelos mais relevantes. Exibindo os modelos, não apenas os desenvolvedores mas também os *stakeholders* podem ter uma idéia do que está sendo criado e em que termos.
- *Criar conteúdo simples*: todos os tipos de modelos do sistema, desde os que retratam requisitos até o relacionados à análise e ao projeto, deve ser feito tão simples quanto possível enquanto ainda cumprirem seus objetivos. A definição de um modelo simples é sugerida em (BECK, 2000) como sendo um modelo que comunica tudo o que é necessário, não contém informação duplicada e utiliza o mínimo possível de elementos para isto.
- *Descrever os modelos de forma simples*: "*conteúdo é mais importante do que a forma de apresentação"*. Apesar de diversas abordagens de modelagem oferecerem inúmeras possibilidades de expressão e diversas notações, a maioria dos modelos criados utiliza um subconjunto limitado dessas notações. Assim, a Modelagem Ágil aconselha os desenvolvedores a utilizarem apenas este subconjunto bem conhecido de elementos suficientes para descrever a maior parte das soluções. Este hábito aumenta a produtividade na criação e na leitura dos modelos, além de eliminar complexidade geralmente desnecessária.
- *Usar as ferramentas mais simples*: equipes adeptas da Modelagem Ágil criam modelos usando lápis e papel em desenhos à mão livre, desenhos em quadros brancos, fotos de esboços criados durante sessões de modelagem. Outras equipes tendem a utilizar ferramentas CASE (*Computer Aided Software Engineering*) extremamente elaboradas para

qualquer atividade de modelagem. Isso faz com que se desacelere o processo de criação de modelos carregando estas ferramentas em memória, utilizando formatos de difícil compartilhamento, utilizando tempo considerável das sessões de modelagem na organização de layout dos modelos. Muitas vezes, porém, um simples desenho poderia ser suficiente para expressar as informações sendo modeladas. Na Modelagem Ágil, ferramentas CASE também são utilizadas, mas apenas quando sua utilidade está justificada – por exemplo, quando será utilizada alguma funcionalidade da ferramenta para a geração automática de código, ou engenharia reversa.

- *Considerar a habilidade de efetuar testes*: durante a modelagem, deve haver uma preocupação constante com a forma pela qual o *software* será testado. Os testes devem ocorrer desde a primeira iteração e com grande freqüência. Algumas abordagens de desenvolvimento promovem esta prática, como XP por exemplo, com seu desenvolvimento orientado a testes.
- *Provar com código*: um modelo é uma abstração da solução sendo desenvolvida. Para saber se o modelo está realmente de acordo com as necessidades, deve-se implementá-lo, submetê-lo aos testes e entregá-lo aos usuários para receber retroalimentação (*feedback*). Esta é a base do desenvolvimento iterativo e incremental.

2.4.3.4 Resumo da Modelagem Ágil

A figura a seguir ilustra os valores, conceitos e práticas da Modelagem Ágil. Existem outros princípios e práticas que complementam os descritos neste trabalho e são chamados de princípios e práticas *suplementares*.

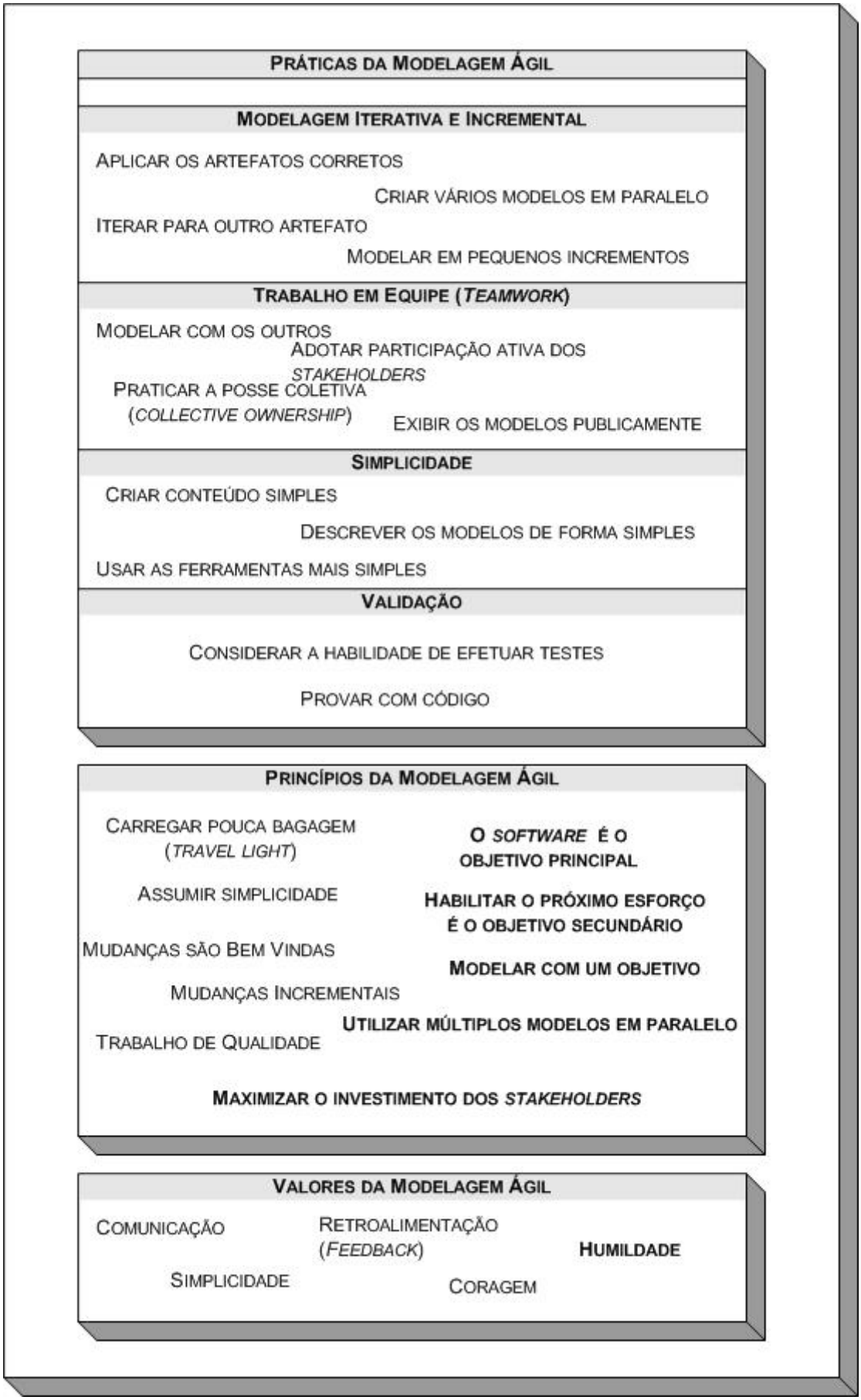


Figura 2.11: Conceitos da Modelagem Ágil

Esta abordagem de desenvolvimento mantém o foco de suas orientações nas atividades de modelagem, e não define processos ou papéis a serem executados em seus projetos. A equipe de desenvolvimento deve seguir algum outro processo para definir e coordenar suas atividades. Esse processo pode ser leve e focado em resultados imediatos como XP ou mais formal e prescritivo como RUP.

Nem todos os processos de desenvolvimento de *software* podem acomodar a Modelagem Ágil. Para concluir se determinado processo comporta esta metodologia, deve-se comparar as definições de tal processo com os valores, princípios e, principalmente, com as práticas da Modelagem Ágil. Se houver discordância com estes conceitos, a metodologia pode ser adotada parcialmente. Porém, os ganhos com a utilização da Modelagem Ágil serão provavelmente reduzidos, uma vez que não se poderá contar com a sinergia entre todas as práticas.

2.5 Integração de Metodologias de Desenvolvimento de Software

Este capítulo descreve três enfoques de desenvolvimento de software:

- *Integração dos Modelos de Capacidade e Maturidade* (*CMMI*): um modelo para melhoria de processos, descrito através de metas e práticas de desenvolvimento;
- *Processo Unificado da Rational* (*RUP*): um *framework* de processo de desenvolvimento que contém orientações sobre aspectos dinâmicos (iterações, fases e marcos) e aspectos estáticos (atividades, artefatos, papéis e fluxos de atividades) de um projeto;
- *Métodos Ágeis*: um conjunto de metodologias que compartilham valores e princípios fundamentais, descritas a partir destes valores e princípios e complementadas por práticas, atividades e estratégias diversas.

Estes três enfoques não são necessariamente incompatíveis. Na verdade, diversos estudos apontam para a existência de pontos de sinergia assim como pontos de incompatibilidade entre os mesmos. A seguir, são sumarizadas algumas análises feitas sobre a utilização destes enfoques em conjunto.

2.5.1 RUP & CMMI

Alguns estudos afirmam que RUP e CMMI são abordagens complementares, já que trazem contribuições em diferentes perspectivas e podem ser utilizadas conjuntamente para ajudar uma organização a alcançar seus objetivos de negócios e de maturidade de processos (TYSON; BROWNSWORD; BROWNSWORD, 2004), (GALLAGHER; BROWNSWORD, 2001).

Como RUP é um processo de desenvolvimento de *software*, é possível avaliá-lo com relação a um modelo de referência para melhoria de processos como o CMMI, um modelo de avaliação de processo já reconhecido na comunidade de engenharia de *software*. Em (MANZONI; PRICE, 2003) citando (FITZGERALD; O'KANE, 1999), afirma-se que é possível estender RUP criando uma instância que inclua práticas chave descritas pelos modelos de CMM, e assim chegar a um modelo de processo que satisfaz a ambas as abordagens.

De acordo com (TYSON; BROWNSWORD; BROWNSWORD, 2004), os conceitos de RUP e do modelo CMMI apresentam a seguinte correspondência:

Tabela 2.2: Correspondência entre Conceitos de RUP e de CMMI

Conceito de RUP	Conceito de CMMI
• *Disciplina*	• Agrupamento de *elementos de processo* logicamente associados
• *Papel*	• Comportamento e responsabilidades de um indivíduo em um projeto
• *Workflow*	• Seqüência de atividades com algum significado, realizada para alcançar algo de valor observável
• *Atividade*	• Unidade de trabalho que um indivíduo pode executar
• *Artefato*, *Modelo* (*Template*)	• *Produtos de Trabalho*
• *Guideline*, *Conceito*, *Orientação de Ferramenta* (*Tool Mentor*)	• *Orientação Informativa* (*Informational Guidance*)

A correspondência apresentada acima pode ser usada para auxiliar a avaliação de RUP com relação aos objetivos e práticas do CMMI.

Em (SMITH, 2000), são apresentadas justificativas para considerar-se que RUP satisfaz todos os critérios necessários para representar um processo aderente ao CMMI Nível 2 e 3.

Em (MANZONI; PRICE, 2001), afirma-se que o estudo de Smith foi feito com foco apenas nos objetivos de cada área de processo e não nas práticas a serem executadas em cada uma destas áreas. O trabalho de Manzoni e Price documenta a comparação entre RUP e CMM, considerando cada uma das práticas recomendadas, e identifica carências no modelo RUP com relação a práticas dos níveis de maturidade 2 e 3 do CMM.

As áreas de RUP onde foram observadas as carências com relação ao CMM são: controle de qualidade, foco no processo da organização, coordenação entre times, treinamento e gerência de sub-contratação de serviços. Em um estudo posterior (MANZONI; PRICE, 2003), os mesmos autores afirmam que a não satisfação das práticas chave por RUP resulta, em geral, da falta de uma descrição sobre como recursos humanos e financeiros adequados são fornecidos para dar seguimento a atividades do processo e da falta de mecanismos que assegurem treinamento adequado para que os desenvolvedores executem suas atividades.

Em (GALLAGHER; BROWNSWORD, 2001), membros do Instituto de Engenharia de *Software* da Universidade de Carnegie Mellon (CMU-SEI), responsável pelo CMMI, relatam uma análise detalhada do suporte fornecido por RUP para o Modelo de Capacidade e Maturidade Integrado de Sistemas/Egenharia de *Software* (CMMI-SE/SW). Neste estudo, investiga-se o suporte fornecido por fluxos de atividades e artefatos de RUP para cada uma das práticas específicas do CMMI-SE/SW.

As áreas de carência de RUP encontradas em (GALLAGHER; BROWNSWORD, 2001) são acentuadas em três áreas de processo: No Planejamento de Projeto, RUP não fornece auxílio para classificar e medir atributos de projetos não relacionados a *software* como trabalho, materiais, máquinas, etc; com relação ao Monitoramento e Controle de Projeto, o *framework* RUP padrão não assegura o endereçamento explícito da gerência de dados; por fim, a área de Gerência de Acordo com Fornecedor está praticamente ignorada, já que RUP não lida explicitamente com a gerência de trabalho vindo de fornecedores externos ao projeto.

Em (MANZONI; PRICE, 2001), apontam-se propostas para estender RUP de forma a satisfazer as áreas de processo do SW-CMM (*Software* CMM) de níveis 2 e 3 de maturidade. Estas propostas descrevem:

- A inclusão de atividades para estimativa de recursos e fundos;
- A inclusão de atividades relacionadas ao controle de qualidade do sistema;
- A inclusão de uma disciplina para gerência de sub-contratação;
- A inclusão de um fluxo de atividades relacionado a treinamento de pessoal;
- A inclusão de atividades relacionadas à melhoria de processos.

As extensões em RUP com o objetivo de atender os requisitos do CMMI Nível 2 descritas em (REITZIG, 2003) são:

- Criar políticas organizacionais que orientem o planejamento e o desempenho de processos do CMMI nível 2;
- Criar padrões e procedimentos detalhados que enderecem o desempenho de atividades do cotidiano;
- Realizar revisões orientadas a processo;
- Gerenciar fornecedores.

A próxima seção analisa a possibilidade de inserção de métodos ágeis em instâncias de RUP.

2.5.2 RUP & Métodos Ágeis

Conforme descrito anteriormente, RUP é um *framework* para processo de desenvolvimento de *software* que pode ser modificado, ajustado e expandido pela organização que o utiliza para acomodar as necessidades, características, restrições e histórico específicos de uma organização, cultura e domínio (KRUCHTEN, 2001).

Ainda em (KRUCHTEN, 2001), observa-se um claro alinhamento da filosofia de RUP com os princípios e práticas da modelagem ágil, ao afirmar-se que:

> "*Um processo não deve ser seguido cegamente, gerando trabalho inútil e produzindo artefatos que são de pouco valor agregado. Ao invés disso, o processo deve ser feito tão leve quanto possível de forma a satisfazer sua missão de produzir rapidamente software previsivelmente de alta qualidade*".

Em (KRUCHTEN, 2001-b), (AMBLER, 2001), (SANTOS, 2002) e (MARTIN, 1998), são descritas alternativas de instanciação de RUP calcadas em princípios ágeis.

(AMBLER, 2002) fornece uma descrição detalhada da relação de todas as práticas da modelagem ágil com as práticas utilizadas nas disciplinas de RUP, indicando as vantagens de utilização das práticas ágeis em cada contexto. Analisando esta relação, percebe-se que o conceito chave a ser utilizado na integração das práticas de modelagem ágil a um processo de desenvolvimento nos moldes de RUP é manter o foco na agilidade do processo.

Algumas decisões em prol da agilidade são feitas sem dificuldades, como a inserção de *stakeholders* em papéis de modelagem, adoção de apenas um subconjunto de artefatos por projeto e criação de protótipos para execução de modelos – boa parte dessas decisões já está incorporada nas próprias práticas de RUP. Outras decisões, contudo, se mostram um tanto mais desafiadoras, como o *ajuste* de padrões e diretrizes (*guidelines*) de forma a eliminar trabalho desnecessário, a adoção de práticas de abandono de modelos intermediários de projeto (*design*) e sincronização de artefatos *apenas* sob demanda.

É importante esclarecer que agilidade, para uma organização de desenvolvimento de *software*, é a habilidade de se adaptar e reagir rápida e apropriadamente a mudanças em seu ambiente e a demandas impostas por este ambiente. Um processo ágil adota e dá suporte a este tipo de adaptabilidade. Assim, o mais importante não é o tamanho do processo ou sua velocidade de entrega, mas sim sua flexibilidade (KRUCHTEN, 2001-b).

RUP possui nove disciplinas, dentre as quais três podem ser claramente aplicadas às praticas de Modelagem Ágil: *Modelagem de Negócio*, *Requisitos* e *Análise e Projeto*. No contexto deste trabalho, a disciplina de *Requisitos* é de especial interesse, já que o processo proposto é para engenharia de requisitos.

A integração da Modelagem Ágil à disciplina de Requisitos de RUP contribui para aumentar a produtividade e facilitar a manutenção dos modelos, através da execução das práticas e princípios da Modelagem Ágil. Esta integração é comentada no próximo capítulo, *Processos e Práticas da Engenharia de Requisitos*, na seção *Métodos Ágeis e a Engenharia de Requisitos*.

De acordo com (AMBLER, 2002), um dos maiores desafios na implantação de práticas ágeis em projetos que utilizam RUP é a cultura da organização à qual pertencem os projetos. Para adotar estratégias ágeis, desenvolvedores devem afastar-se da forma prescritiva de desenvolvimento, freqüentemente calcada em produção de documentação, para que possam se aproximar de valores ágeis como simplicidade e maximização do investimento dos *stakeholders*, mantendo foco no sistema a ser produzido e tratando todas as outras atividades como meios para a produção de código que atenda às necessidades do cliente.

A próxima área de investigação da utilização conjunta de abordagens de desenvolvimento de *software* neste trabalho é a avaliação de processos que utilizam métodos ágeis comparando-os aos modelos do CMMI.

2.5.3 CMMI e Métodos Ágeis

Segundo (PAULK, 2001), os modelos do SW-CMM consideram tanto questões relacionadas à implementação de processos eficazes e eficientes quanto a melhoria sistemática de processos.

Já os métodos ágeis, em geral representam conjuntos de práticas específicas cujo contexto ideal de aplicação é o de projetos de tamanho

pequeno ou médio, com times localizados em um mesmo local e cujos requisitos estão sofrendo alterações continuamente.

Diversos estudos (JEFFRIES, 2000), (REIFER, 2003), (PAULK, 2001) afirmam que estas duas categorias de métodos (CMMI ou CMM e métodos ágeis) podem ser empregadas conjuntamente, criando uma sinergia que possibilita que as organizações que os utilizam se beneficiem das vantagens trazidas por ambos.

Em (JEFFRIES, 2000), Ron Jeffries, um dos idealizadores da Aliança Ágil, afirma que XP, um dos métodos ágeis considerados neste trabalho, possui algumas características em comum com todos os níveis de maturidade do CMM, inclusive o nível 5. Porém, Jeffries faz a ressalva de que seria necessário muito mais documentação e comprovações do que o sugerido por XP para considerar um projeto que utiliza esta abordagem como aderente a qualquer nível de maturidade do CMM. Ainda neste estudo, comenta-se que as práticas de XP, claramente definidas e documentadas, representariam uma forma de alcançar o objetivo genérico de *Institucionalizar um Processo Gerenciado*.

Em (REIFER, 2003), sugere-se que deveria haver um empenho do Instituto de Engenharia de *Software* da Universidade de Carnegie Mellon (CMU/SEI), órgão que elabora e divulga os requisitos do CMMI, no sentido de disponibilizar orientação para organizações interessadas em adotar os modelos do CMMI com a inclusão de práticas dos métodos ágeis. Tal empenho facilitaria a transição para abordagens como XP e ainda poderia trazer a organizações que já comprovaram ter atingido níveis de maturidade altos as vantagens da agilidade, como por exemplo o aumento da produtividade.

Um esforço neste sentido é o artigo de Mark Paulk (PAULK, 2001), membro da equipe do SEI, analisando as contribuições que XP pode trazer a organizações que procuram alcançar os objetivos do modelo SW-CMM. Neste artigo, Paulk analisa cada uma das áreas de processo do SW-CMM e discute práticas e mecanismos de XP que dão suporte àquela prática. A tabela abaixo, baseada na tabela de satisfação de áreas chave de processo apresentada por Paulk, resume a discussão do suporte de XP a áreas de processo do nível 2 de maturidade.

Tabela 2.3: Satisfação das áreas chave de processo do SW-CMM por XP (PAULK, 2001)

Nível Mat.	**Área Chavede Processo**	**Índice de Satisfação**	**Mecanismos de Suporte**
2	Gerência de Requisitos	Satisfaz Amplamente	Estórias do Usuário Cliente Presente Integração Contínua Priorização de estórias pelo cliente
2	Planejamento de Projeto de *Software*	Satisfaz Amplamente	Jogo do Planejamento Pequenas *Releases* Estimativas feitas pelos desenvolvedores (compromisso) Priorização de estórias pelo cliente
2	Acompanhamento e Supervisão de Projeto de *Software*	Satisfaz Amplamente	Acompanhamento de medidas (ex: velocidade de projeto)

Nível Mat.	Área Chavede Processo	Índice de Satisfação	Mecanismos de Suporte
2	Gerência de Subcontratação de *Software*	Não satisfaz	Não há suporte A ocorrência de sub-contratação de *software* em projetos usando XP é provavelmente uma raridade
2	Garantia de Qualidade de *Software*	Satisfaz Parcialmente	Programação em Pares (embora em times maiores exista a necessidade de uma maior formalização das métricas de qualidade)
2	Gerência de Configuração	Satisfaz Parcialmente	Propriedade Coletiva de Código (arriscada em projetos maiores) Pequenas *Releases* Integração Contínua
3	Foco no Processo Organizacional	Satisfaz Parcialmente	Escolha da forma de adoção de XP (todas as práticas ou uma de cada vez?)
3	Definição do Processo Organizacional	Satisfaz Parcialmente	Parcialmente endereçadas pelos diversos livros, artigos, cursos e *web sites* associados a XP.
3	Treinamento da Organização	Não satisfaz	Parcialmente endereçadas pelos diversos livros, artigos, cursos e *web sites* associados a XP.
3	Gerência de Software Integrada	Não satisfaz	Recursos organizacionais estão fora do escopo de XP.
3	Engenharia de Produto de Software	Satisfaz Amplamente	Metáfora Projeto Simples Refatoração Padrões de Código Testes Unitários e Funcionais
3	Coordenação Inter-Grupos	Satisfaz Amplamente	Cliente Presente Programação em Pares
3	Revisões por Pares	Satisfaz Amplamente	Revisão por Pares

De acordo com (PAULK, 2001), o principal elemento ausente em XP e fundamental para o SW-CMM é o conceito de *instucionalização*. Ou seja, em XP não há a formalização do processo seguido pela organização, prática que o SW-CMM considera necessária. Outro estudo (JEFFRIES, 2000) afirma que as práticas de XP, claramente definidas e documentadas, representariam uma forma de alcançar o objetivo genérico de *Institucionalizar um Processo Gerenciado*.

Ainda na comparação feita por Paulk (PAULK, 2001), aponta-se que projetos maiores oferecem dificuldades consideráveis para a aplicação de algumas práticas de XP, como por exemplo a propriedade coletiva do código.

Nas análises estudadas sobre métodos ágeis e CMM ou CMMI (JEFFRIES, 2000), (REIFER, 2003), (PAULK, 2001), a principal contribuição dos métodos ágeis parece ser em sua filosofia. Ou seja, mesmo que não se adote especificamente as técnicas e atividades propostas pelos métodos ágeis, organizações que buscam a conformidade com CMMI podem e devem

incorporar diversas práticas ágeis, tais como a procura pela simplicidade e a retroalimentação (*feedback*).

3 PROCESSOS E PRÁTICAS DA ENGENHARIA DE REQUISITOS

Esta seção define a engenharia de requisitos, os conceitos relacionados a esta disciplina, sua importância para projetos de desenvolvimento de software, problemas clássicos e possíveis soluções.

São apresentados, ainda, recursos e técnicas do Processo Unificado de Rational (RUP) e dos métodos ágeis com relação à engenharia de requisitos. Por fim, são descritas as metas e práticas das áreas de processo de Gerência de Requisitos e de Desenvolvimento de Requisitos do CMMI.

3.1 Requisitos: Conceitos e Definições

Para compreender processos e práticas de engenharia de requisitos para projetos de desenvolvimento de software, é preciso conhecer a definição de requisito, seus níveis, tipos e classificações e dependências. Estes assuntos são abordados nas seções seguintes.

3.1.1 Definição de Requisito

Os requisitos são o ponto de partida para toda a definição do sistema e, conseqüentemente, são fatores decisivos no desenvolvimento do produto final. A literatura relacionada à engenharia de requisitos oferece diversas definições do conceito de *requisito*.

De acordo com (BITTNER; SPENCE, 2002), um requisito é:

> *Uma condição ou capacidade com a qual o sistema deve estar em conformidade.*

Já em (SOMMERVILLE, 2001), a definição encontrada é:

> *Uma especificação do que deve ser implementado ou uma restrição de algum tipo do sistema.*

Por fim, a definição padronizada pela IEEE em seus *Padrões, Guidelines e Exemplos sobre Engenharia de Requisitos de Sistemas e de Software* é a seguinte (DORFMAN; THAYER, 1990):

1. *Uma condição ou capacidade necessária a um usuário para resolver um problema ou alcançar um objetivo.*
2. *Uma condição ou capacidade que deve ser alcançada ou possuída por um sistema ou por um componente de sistema para satisfazer um contrato, padrão, especificação ou outros documentos formalmente expostos.*
3. *Uma representação documentada de uma condição ou capacidade como a dos itens 1 ou 2.*

Este trabalho segue todas estas definições, mas utiliza a última (DORFMAN; THAYER, 1990) como referência, por considerá-la a mais abrangente. Contudo, o termo *usuário* deve ser substituído por *interessado* (*stakeholder*), já que o cliente e suas diversas equipes envolvidas na definição e utilização do sistema também devem fornecer requisitos a este sistema.

Partindo destas definições, alguns exemplos de requisitos seriam:

- "O sistema deve fornecer informações sobre todas as ações executadas por seus usuários em qualquer período de tempo."
- "O sistema deve estar integrado ao sistema bancário XXY utilizado pela organização."
- "O sistema deve oferecer ao cliente a compra menos custosa que satisfaça seus parâmetros de definição do produto."
- "Precisamos diminuir as vendas que resultam em fraude no pagamento."
- "O sistema deve seguir um processo de desenvolvimento baseado em RUP."
- "O sistema deve gerar relatórios sobre vendas por período, evidenciando os parâmetros de avaliação dos envolvidos."

3.1.2 Níveis de Requisitos

3.1.2.1 Necessidades, Características e Requisitos de Software

Requisitos são de natureza variável, podem ser uma descrição de funcionalidade no nível do usuário, uma especificação detalhada do comportamento esperado de um sistema, uma propriedade genérica de um sistema, uma restrição técnica do sistema, uma restrição no processo de desenvolvimento, informações sobre como realizar determinado cálculo, etc. (SOMMERVILLE, 2001).

Os requisitos podem ser classificados em diferentes níveis, de forma que o requisito em um nível dá origem a um ou mais requisitos no nível seguinte (*requisitos derivados*). O Processo Unificado da Rational (LEFFINGWELL; WIDRIG, 2000), que divide os requisitos em:

- *Necessidades*: reflexões de problemas de negócio, pessoais ou operacionais que devem ser endereçadas para justificar o desenvolvimento de um novo sistema.
- *Características*: serviços observáveis externamente através dos quais o sistema satisfaz uma ou mais necessidades dos *stakeholders*;
- *Requisitos de Software*:
    - Requisitos que especificam como o sistema interage com o contexto à sua volta (**Requisitos Funcionais**);
    - Requisitos que expressam atributos de qualidade da solução (**Requisitos Não Funcionais**)

A *necessidade* situa-se no espaço do problema, ou seja, representa uma dificuldade que deve ser resolvida com a utilização do sistema. As necessidades estão no primeiro nível de requisitos em um projeto, dando origem a todos os outros requisitos. Exemplos de necessidades são:

- "Precisamos de mais agilidade no processamento de vendas."
- "O check-in deve envolver menos burocracia."

- “O sistema deve fornecer informações sobre todas as ações executadas por seus usuários em qualquer período de tempo.”
- “O sistema deve ser o mais automatizado possível.”
- “Precisamos diminuir o número de compras com fraude no pagamento.”
- “Precisamos identificar nossos melhores clientes.”

A *característica* é uma propriedade ou função do sistema. Assim, está no *espaço da solução*, já que descreve como o sistema deve satisfazer às necessidades (BITTNER; SPENCE, 2002). Alguns exemplos de características estão listados abaixo.

- “A compra de produtos deve ser completamente automatizada, da consulta ao pagamento.”
- “O sistema deve permitir a requisição, sob demanda, de um relatório com todas as ações executadas por seus usuários.”

Os *requisitos de software* são funcionalidades do sistema ou qualidades do mesmo. Enquanto as características são orientadas à capacidade do sistema, os requisitos de *software* são levam em conta a interação do usuário com o sistema. Exemplos de requisitos de *software* são:

- “Para realizar um compra, o cliente consulta os produtos disponíveis, escolhe o produto desejado, informa seus dados de endereço e pagamento...”
- “O processamento de uma transação de compra leva no máximo 10 segundos para ser finalizado.”

A figura abaixo é a representação clássica utilizada por RUP para ilustrar os níveis de requisitos. Note-se que, em um projeto típico, o número de necessidades é bem menor que o de características, que por suas vez são menos numerosas que os requisitos de software.

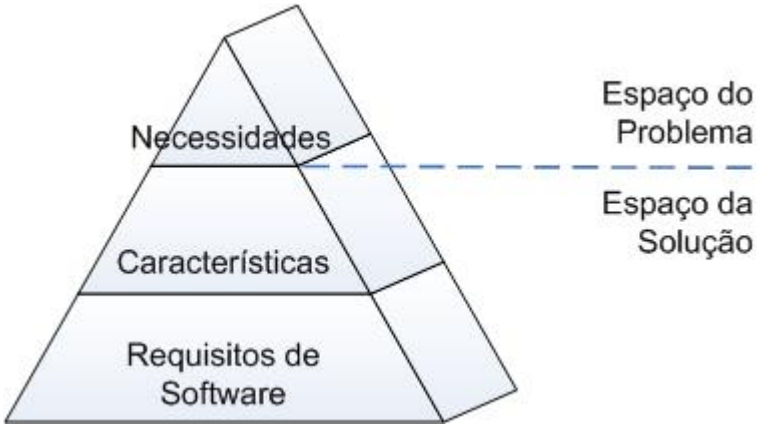


Figura 3.1: Níveis de Requisitos de RUP, adaptada de (RATIONAL UNIVERSITY, 2002-b)

3.1.2.2 Rastreabilidade

Os requisitos de um sistema relacionam-se entre si e com os demais artefatos do projeto, até o produto final, ou seja, até o sistema de *software* construído. Estas relações implicam em dependências entre os requisitos e artefatos, de tal forma que a alteração de um requisito pode resultar na necessidade mudança em outros requisitos ou artefatos dependentes.

Assim, para manter a coerência dos requisitos entre si e com o produto final, é preciso documentar as relações entre estes itens. A relação entre dois ou mais produtos do processo de desenvolvimento é chamada de *rastreabilidade* (IEEE, 1990).

A rastreabilidade pode partir de requisitos de um nível mais abstrato para o produto final (*forward traceability*). Pode, ainda, partir do produto final para os requisitos que o originaram (*backwards traceability*) (WIERINGA, 1995). A presença destes dois tipos de rastreabilidade simultaneamente é chamada de rastreabilidade *bidirecional* (SOFTWARE ENGINEERING INSTITUTE, 2005-b).

De acordo com as definições de níveis de requisitos apresentadas anteriormente, uma necessidade é satisfeita por uma ou mais características; uma característica é implementada por um ou mais requisitos de software; um requisito de software formaliza o funcionamento ou as propriedades de uma ou mais características. Estas relações configuram a *rastreabilidade vertical*, ou seja, relações entre requisitos de níveis diferentes.

Requisitos também podem apresentar dependências dentro de um mesmo nível. Por exemplo, a característica “Os relatórios do sistema são sempre agrupados por período” está relacionada à característica “O arquivo XXX possui um exemplo de relatório de atividades de usuários”. Relações entre requisitos de mesmo nível caracterizam a *rastreabilidade horizontal*.

A definição de rastreabilidade (IEEE, 1990) e dos conceitos de rastreabilidade horizontal e vertical (SOFTWARE ENGINEERING INSTITUTE, 2005-b) não se restringem a requisitos; tais definições abrangem os demais produtos do processo de desenvolvimento como artefatos de teste, modelos de implementação e planos de projeto.

Note-se que a rastreabilidade progressiva (*forward*), a regressiva (*backwards*) e, conseqüentemente, a bidirecional, são exemplos de rastreabilidade vertical.

A figura a seguir ilustra os conceitos de rastreabilidade apresentados acima.

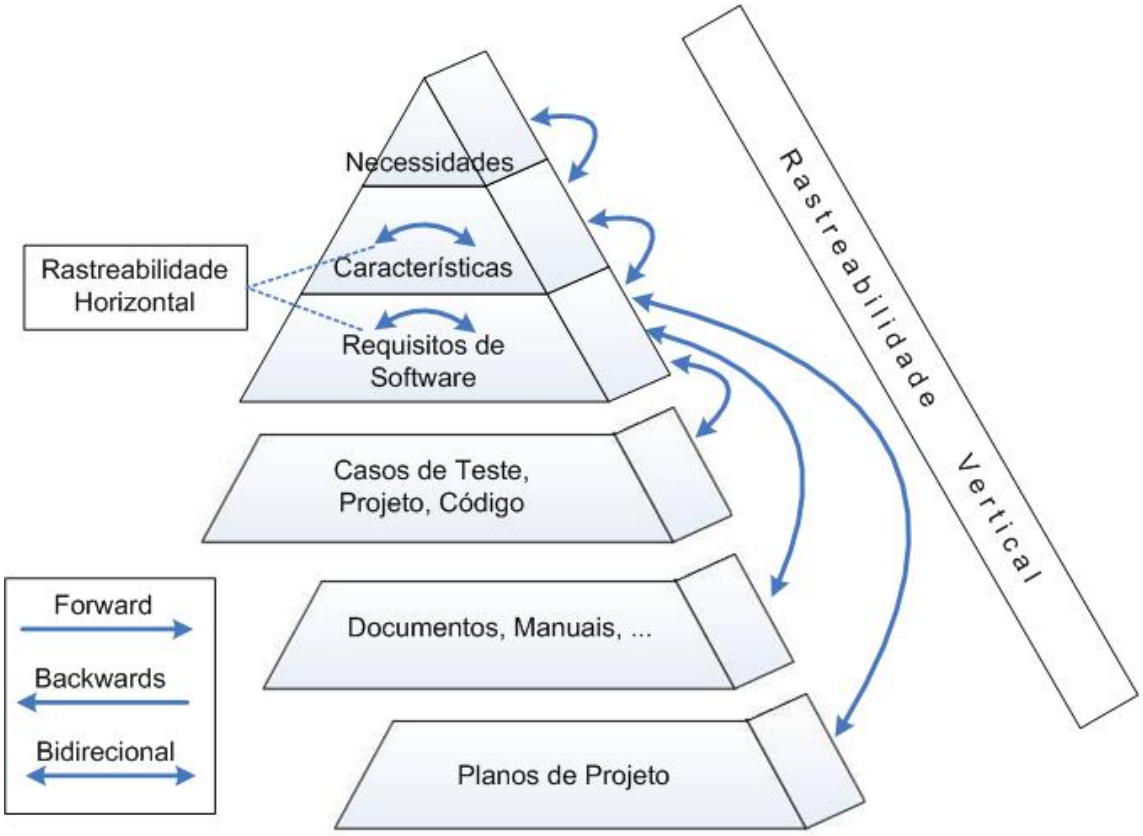


Figura 3.2: Rastreabilidade de Requisitos, adaptada de (RATIONAL UNIVERSITY, 2002-b)

A rastreabilidade é um recurso importante da gerência de requisitos, já que:

- Possibilita a descoberta de requisitos fonte que ainda não estão cobertos pelos requisitos de nível mais baixo ou mesmo por componentes do sistema implementado. Por exemplo, se uma necessidade não está ligada (rastreada) a nenhuma característica, isto significa que não há nenhum componente do sistema sendo desenvolvido para satisfazer esta necessidade. Isso significa que atividades de análise de requisitos devem ser executadas para que se definam os requisitos que satisfazem à necessidade em questão.
- Possibilita a descoberta de requisitos que não possuem origem nas necessidades dos *stakeholders*. Estes requisitos são chamados de *goldplating* e devem ser excluídos do sistema, pois representam um custo adicional que não tem como foco as necessidades do cliente.
- Facilita a análise de impacto de mudanças nos requisitos, evidenciando os requisitos e artefatos do projeto que devem ser revisados quando um requisito é alterado.
- Auxilia nas decisões tomadas com relação a requisições de mudança, possibilitando verificar se uma mudança em um requisito ou artefato do projeto não interfere na satisfação de uma necessidade fonte.

Documentar a rastreabilidade entre requisitos e produtos de trabalho do projeto significa, simplesmente, registrar quais destes produtos possuem dependências entre si. Esta documentação é feita por meio de matrizes de rastreabilidade, que evidenciam a dependência de grupos de requisitos. Ferramentas de auxílio ao desenvolvimento de software podem oferecer

funcionalidades avançadas relacionadas a matrizes de rastreabilidade como a geração automática de matrizes, a geração de árvores de dependência e o destaque de todos os requisitos afetados por mudanças em um determinado requisito, facilitando a análise de impacto da mudança.

3.1.2.3 Restrições

"As restrições são limitações no projeto do sistema ou no processo utilizado para construí-lo manifestadas pelos stakeholders" (RATIONAL UNIVERSITY, 2002-b). Restrições no processo incluem restrições com relação à verificação (conformidade do sistema com relação aos requisitos definidos) e à validação (conformidade do sistema com relação às necessidades do cliente) (SOFTWARE ENGINEERING INSTITUTE, 2002).

Algumas restrições resultam em requisitos. Outras, afetam planos de projeto e recursos. De uma forma ou de outra, uma restrição é uma limitação no grau de liberdade que se tem no desenvolvimento de uma solução (LEFFINGWELL; WIDRIG, 2000).

Exemplos de restrições são:

- "O sistema deve ser desenvolvido utilizando um processo que satisfaça, comprovadamente, os requisitos do CMMI".
- "O sistema deve ser construído utilizando o Framework Microsoft .NET e linguagem C#".
- "A base de dados deve ser Oracle 9i".
- "A verificação do sistema deve incluir procedimentos automatizados para que estes possam ser repetidos a cada nova versão."
- "Cada ponto de avaliação do sistema deve incluir um *workshop* no qual os usuários do sistema terão acesso a um protótipo funcional para aprovação das funcionalidades já implementadas."

Ao identificar uma restrição, é importante compreender as justificativas para a mesma. As justificativas evidenciam a perspectiva na qual a restrição deve ser considerada e possibilitam identificar quando a restrição não se aplica ao sistema (LEFFINGWELL; WIDRIG, 2000).

3.1.2.4 Requisitos do Cliente e Requisitos do Produto

A descrição de metas e práticas dos modelos do CMMI utiliza os seguintes conceitos de requisito (ZUBROW, 2004):

- *Requisitos do Cliente*: requisitos do sistema manifestados por seus *stakeholders* (necessidades, expectativas, restrições definidas pelo cliente, usuários finais, gerentes e demais interessados).
- *Requisitos de Produto*: requisitos do sistema visíveis externamente.
- *Requisitos de Componente de Produto*: requisitos internos do sistema.

As definições de *produto* e *componente de produto*, neste contexto, são (SOFTWARE ENGINEERING INSTITUTE, 2002):

- *Produto*: qualquer artefato ou serviço resultante de um processo e que será entregue ao cliente ou usuário final. Exemplos destes artefatos são arquivos, documentos, partes de produto, processos (ex: processo de treinamento), especificações, etc.

- *Componente de Produto*: parte de um produto. Diversos componentes de produto são integrados para resultar em um produto completo. Um componente possui requisitos, projeto (*design*) e implementação.

O conceito de requisito de produto é utilizado de uma forma generalizada em (SOFTWARE ENGINEERING INSTITUTE, 2002), referenciando requisitos de produto *e* de componente de produto. Esta generalização é utilizada neste trabalho.

Note-se que requisitos de cliente e de produto também representam níveis de requisitos e se sobrepõem aos demais níveis definidos anteriormente da seguinte forma:

- Requisitos do cliente são necessidades e características;
- Requisitos do produto são requisitos de software.

A figura abaixo representa a unificação destas definições.

Figura 3.3: Níveis de Requisitos RUP e CMMI

3.1.3 Tipos de Requisitos de Software

Os requisitos de software são, em geral, bastante numerosos e, por isso, devem ser organizados e agrupados para facilitar sua gerência e manutenção. O modelo **FURPS** de classificação de requisitos, proposto em (GRADY, 1992), define os tipos de requisito a seguir:

- *Requisitos de Funcionalidade (**F**unctionality):*
  Requisitos orientados a ações, ao comportamento do sistema, à interação entre o usuário e o sistema. Representam, geralmente, o maior número de requisitos do sistema.
- *Requisitos de Usabilidade (**U**sability):*
  Consideram a experiência dos usuários com computadores e com o domínio do sistema; facilidade de treinamento; freqüência de uso do sistema; motivações de uso do sistema pelos usuários; estética; consistência da interface com o usuário; ajuda (*help*) on-line e sensível ao contexto; assistentes e agentes automatizados; documentação do usuário e manuais de treinamento.
- *Requisitos de Confiabilidade (**R**eliability)*

Freqüência e severidade de falhas; possibilidade de recuperação; previsibilidade; precisão e tempo médio entre falhas (MTBF – *mean time between failure*).

- *Requisitos de Desempenho (**P**erformance)*

  Velocidade; eficiência; precisão; capacidade de carga (*throughput*); tempo de resposta; tempo de recuperação; utilização de recursos operacionais.

- *Requisitos de Suporte (**S**upportability):*

  Possibilidade de teste; possibilidade de extensão do software; adaptabilidade; manutenibilidade; compatibilidade; flexibilidade de configuração e instalação; internacionalização.

Os tipos de requisitos mencionados acima ajudam na organização dos requisitos, facilitando sua gerência. Em (IBM RATIONAL CORPORATION, 2003), este modelo foi complementado, criando o FURPS+, com a inclusão de:

- *Restrições ou Requisitos de Projeto (design)*

  Especificam ou restringem o projeto (*design*) de um sistema.

- *Requisitos de Implementação*

  Especificam ou restringem o código ou construção de um sistema: padrões requeridos; linguagem de implementação; políticas para a integridade de bases de dados; limitação de recursos; ambientes operacionais.

- *Requisitos de Interface*

  Especificam um item externo com o qual o sistema deve interagir; restrições nos formatos, tempos de resposta ou outros fatores utilizados nesta interação.

- *Requisitos Físicos*

  Especificam características físicas que um sistema deve possuir, como material, forma, tamanho ou peso. Podem ser utilizados para representar requisitos de hardware.

A lista dos requisitos FURPS+ pode ser dividida em duas categorias:

- *Requisitos Funcionais (**F**)*: os requisitos funcionais especificam ações que um sistema deve ser capaz de desempenhar, sem a consideração de restrições físicas.
- *Requisitos Não Funcionais (**URPS+**)*: requisitos que não descrevem funcionalidade, representando apenas atributos do sistema.

3.1.4 Qualidade do Conjunto de Requisitos do Sistema

Em (IEEE, 1998), os atributos que conferem qualidade ao conjunto de requisitos de um sistema são definidos como sendo os seguintes:

- *Correto*: Todos os requisitos são algo que o sistema realmente deve fazer. Esta qualidade é verificável apenas através de revisão por especialistas.
- *Completo*: O conjunto inclui todos os requisitos significativos: funcionalidades, desempenho, restrições técnicas, atributos e interfaces com sistemas externos. Inclui, ainda, todas as respostas possíveis a entradas válidas e inválidas.

- *Consistente*: Não existe conflito entre os requisitos, considerando-se todos os níveis (necessidades, características, requisitos de software).
- *Não ambíguo*: Cada requisito possui somente uma interpretação.
- *Verificável*: Cada requisito pode ser testado e comprovado a um custo efetivo, automaticamente (máquina) ou manualmente (pessoa). Exemplos de requisitos não verificáveis são: “a interface gráfica do sistema deve exibir cores agradáveis aos olhos”, “o sistema deve funcionar em qualquer navegador”.
- *Ordenável (ranked)*: Cada requisitos deve ser ordenável por importância para o cliente e/ou estabilidade.
- *Modificável*: Mudanças nos requisitos não devem afetar a estrutura e estilo do conjunto de requisitos, ou seja, mudanças devem poder ser feitas com facilidade.
- *Compreensível*: Cada requisito deve estar claro para clientes e desenvolvedores.
- *Rastreável*: Cada requisito deve ser identificado e relacionado aos demais. Além do nome, os requisitos geralmente possuem um identificador numérico ou alfanumérico, único dentro do conjunto de requisitos do sistema.

3.2 Engenharia de Requisitos: Definições e Problemas Clássicos

As seções a seguir descrevem a abrangência da engenharia de requisitos em um projeto, os principais problemas relacionados e as características comuns em soluções encontradas para resolver ou minimizar estes problemas.

3.2.1 Definição da Engenharia de Requisitos

A engenharia de requisitos pode ser descrita como "*o ramo da engenharia de software que se preocupa com os objetivos, funções e restrições dos sistemas de software*" (ZAVE, 1997).

A engenharia de requisitos cobre todas as atividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos para um sistema de *software*. As técnicas da engenharia de requisitos devem garantir que os requisitos do sistema sejam completos, consistentes e relevantes (SOMMERVILLE, 2001).

Em (POHL, 1993), citado por (MACAYLAY, 1997), apresenta-se a seguinte definição de Engenharia de Requisitos:

> *A Engenharia de Requisitos pode ser definida como o processo sistemático de desenvolvimento de requisitos através de um processo iterativo cooperativo de análise do problema, documentação das observações resultantes em variados formatos de representação e a verificação da precisão do ganho compreendido.*

Todas as definições apresentadas acima são compatíveis com este trabalho e com o processo de engenharia de requisitos aqui descrito. Apesar de fornecer as definições acima, a literatura relacionada a requisitos evidencia uma falta de consenso na definição de termos como *engenharia de requisitos*, *gerência de requisitos* ou *análise de requisitos* (WIEGERS, 2000). Estes termos podem aparecer como sinônimos, ou como componentes uns dos outros,

dependendo da referência. Este trabalho adota a definição utilizada por (WIEGERS, 2000), ilustrada na figura abaixo, que é compatível com a organização dos modelos CMMI (SOFTWARE ENGINEERING INSTITUTE, 2002).

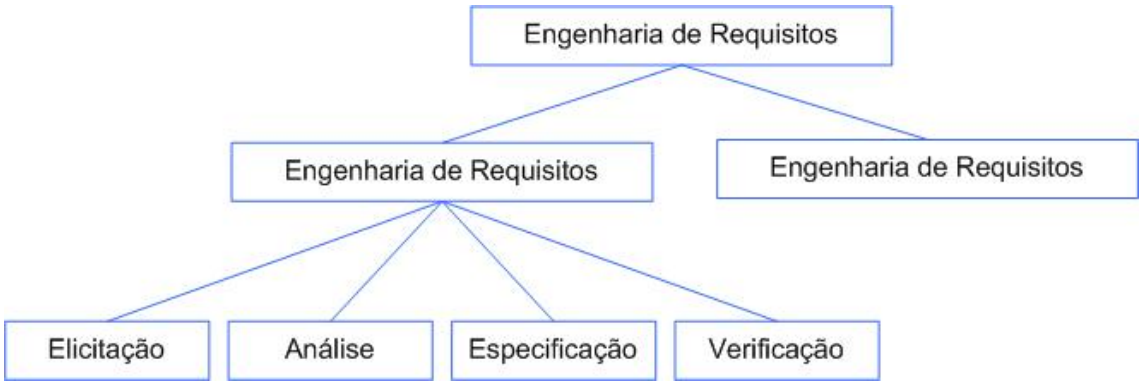


Figura 3.4: Composição da disciplina de Engenharia de Requisitos

[Traduzida de (WIEGERS, 2000)]

De acordo com a definição adotada, a *Engenharia de Requisitos* é dividida em *Desenvolvimento de Requisitos* e *Gerência de Requisitos*. O Desenvolvimento de Requisitos preocupa-se com a descoberta, busca da qualidade (correção, completude, consistência, possibilidade de verificação, ordenação e rastreamento, facilidade de modificação e clareza), detalhamento, documentação, revisão e verificação dos requisitos do sistema. Estas preocupações são endereçadas pelas sub-divisões de *Elicitação*, *Análise*, *Especificação* e *Verificação*. Já a Gerência de Requisitos compreende a gerência de mudanças e o acompanhamento do desenvolvimento e implementação dos requisitos.

Uma lista das principais causas de risco de falha em projetos de *software*, apresentada em (BOEHM, 1991), menciona fatores relacionados à engenharia de requisitos da terceira à sexta colocação:

3. Desenvolvimento de funções e propriedades erradas;
4. Desenvolvimento da interface com o usuário errada;
5. *Gold-plating* (desenvolvimento de requisitos supérfluos);
6. Fluxo constante de mudanças nos requisitos.

Uma lista semelhante, apresentada em (SOFTWARE ENGINEERING INSTITUTE, 2002), cita *requisitos incertos* como a principal fonte de risco em projetos de desenvolvimento de software.

Tamanha relevância atribuída aos resultados da engenharia de requisitos se explica pelo fato de que, caso os requisitos do sistema não sejam compreendidos corretamente, o tempo e os recursos financeiros gastos com a correção do sistema podem fazer com que este seja entregue com atraso e com custo maior do que o originalmente estimado. Caso esta correção não seja feita, o cliente e os usuários finais podem não ficar satisfeitos com o sistema entregue e podem até mesmo deixar de usá-lo (SOMMERVILLE, 2001).

Para que a engenharia de requisitos seja propriamente desenvolvida em um projeto de software, o processo utilizado no desenvolvimento deve ser descrito claramente, com todas as suas fases, atividades e *produtos de trabalho*, ou seja, entradas e saídas do processo (MACAYLAY, 1997).

3.2.2 Gerência de Requisitos

É um processo que estabelece e mantém acordos entre o cliente e a equipe do projeto sobre a evolução dos requisitos (LEFFINGWEL; WIDRIG, 2000).

A gerência de requisitos monitora o desenvolvimento e implementação dos requisitos, registrando seus atributos, status e dependências, com o objetivo de controlar o andamento e as mudanças realizadas. Um conceito da disciplina de Planejamento que está relacionado à gerência de requisitos é a gerência de escopo.

3.2.2.1 Gerência de Escopo

O conjunto de requisitos que serão entregues ao final do projeto de desenvolvimento representa o *escopo* do sistema. Ao longo de um projeto de desenvolvimento de software, os requisitos do sistema são identificados e documentados mediante a utilização de diversas técnicas de elicitação, análise e especificação de requisitos. Contudo, nem todos os requisitos encontrados fazem parte do escopo sistema. A escolha dos requisitos que devem fazer parte do sistema é chamada de *gerência de escopo*.

A gerência de escopo leva em conta fatores como: a importância de cada requisito para os clientes; a relevância técnica de cada requisitos, ou seja, o quanto cada requisito é importante para a definição da arquitetura do sistema; a dependência entre requisitos; as restrições do projeto em termos de orçamento, recursos e prazos. Estes fatores são levados em conta para decidir que requisitos fazem parte do escopo do sistema.

3.2.2.2 Gerência de Mudanças

O conjunto de requisitos que fazem parte do escopo de um sistema muda ao longo de seu desenvolvimento. Alguns fatores responsáveis por estas mudanças são:

- Mudanças no negócio, resultando em alterações nas necessidades do cliente que trazem novos requisitos ou invalidam requisitos formalizados anteriormente.
- Aumento da complexidade de requisitos, identificado após investigação detalhada do requisito.
- Alteração estrutural no requisito necessária para melhor integrá-lo ao sistema ou porque sua estrutura original não configurava uma solução correta.
- Evolução da percepção do sistema pelos fornecedores de requisitos, após a visualização de protótipos ou a execução dos primeiros testes de validação.
- A descoberta de falhas no entendimento e na especificação de requisitos e a conseqüente necessidade de correção.

As mudanças nos requisitos demandam um replanejamento do projeto, possivelmente necessitando de negociações envolvendo variáveis como custo, prazos ou escopo do sistema. Se uma destas variáveis é alterada, as outras sofrem conseqüências. Se o escopo aumenta (por causa do levantamento de novos requisitos, por exemplo), o tempo pode ter que aumentar e o orçamento provavelmente também aumentará. Por outro lado, se o orçamento, recursos e

tempo não podem variar, então o escopo deve ser diminuído, deixando alguns requisitos de menor prioridade fora do sistema.

O papel da gerência de requisitos, neste contexto, é o de oferecer subsídios como a estimativa de esforço, estratégias de priorização e análise de impacto de mudanças sobre os requisitos. A análise de impacto utiliza fortemente as relações entre requisitos, ou seja, a rastreabilidade documentada entre os requisitos (ver seção sobre *Rastreabilidade*).

3.2.3 Problemas Clássicos e Soluções Adotadas

A engenharia de requisitos possui alguns problemas clássicos, observados na maioria dos projetos de software (SOMMERVILLE, 1992), (MACAYLAY, 1997). Estes problemas são comentados abaixo, assim como algumas soluções ou estratégias de contorno:

- *Constantes solicitações de mudança*:

  Conforme descrito na seção anterior, o conjunto de requisitos sobre mudanças ao longo do projeto, por uma série de razões. Assim, é imprescindível, para qualquer processo de desenvolvimento de *software*, que seja definido um mecanismo de tratamento de mudanças nos requisitos. A falta deste resulta em diversos problemas, como por exemplo a entrega de um sistema que não satisfaz as necessidades de seus clientes e usuários ou a falha na entrega do sistema devido a mudanças constantes.

  O mecanismo de tratamento de mudanças deve identificar quais são os pontos de origem de solicitações de mudanças, como estas mudanças devem ser registradas, como analisar o impacto de sua execução, quais os critérios de aprovação de mudanças e como estas devem ser inseridas no planejamento do sistema.

- *Falta de consenso entre stakeholders*:

  Sistemas de *software* geralmente têm uma comunidade heterogênea de usuários, com requisitos e prioridades diferentes e às vezes até mesmo conflitantes. Além disso, os responsáveis pelo orçamento do sistema raramente são os usuários do mesmo. Estes responsáveis impõem requisitos ao sistema que são decorrentes de restrições organizacionais ou financeiras as quais, freqüentemente, entram em conflito com as necessidades operacionais dos usuários. Os requisitos finais do sistema são, inevitavelmente, um acordo entre as partes.

  Para chegar a este acordo, é fundamental que se identifiquem todos os *stakeholders* relevantes para o projeto, incluindo responsáveis pela operação do sistema, equipes envolvidas na visão estratégica da empresa, clientes encarregados de aprovar o orçamento e qualquer outro envolvido cuja participação na definição do sistema seja importante para a aceitação final, utilização e bom funcionamento do sistema.

- *Pouco conhecimento do domínio do problema*:

  Em diversos sistemas, os desenvolvedores não possuem conhecimento sobre o domínio do problema que o sistema deve resolver. Nestes casos, os desevolvedores devem adquirir e evoluir seu conhecimento sobre o domínio e as áreas de atuação dos usuários e sobre as opções e as questões tecnológicas mais relevantes envolvidas na solução. Da mesma forma, os *stakeholders* devem acompanhar a elaboração do novo sistema, evoluindo

sua percepção sobre o mesmo, e sobre a melhor maneira de endereçar os seus problemas.

Estas necessidades apontam para duas estratégias de solução: uma relacionada ao processo utilizada e outra relacionada à equipe de desenvolvimento. Com relação ao processo, a utilização de protótipos e desenvolvimento iterativo e incremental, onde cada iteração contribui para uma crescente compreensão do sistema, e da realidade de negócios que o cerca, é uma linha de ação promissora. Com relação à equipe, identifica-se a necessidade de contratação de pessoal experiente, consultores proficientes no negócio, tutores versados no domínio e no processo de desenvolvimento do sistema.

- *Pouca qualidade da documentação e comunicação*:

Se a documentação dos requisitos não é de boa qualidade e a comunicação entre os *stakeholders*, os analistas e os responsáveis pela implementação do sistema é deficitária, as chances de que o sistema final realmente satisfaça as necessidades dos *stakeholders* são drasticamente prejudicadas.

Em (MACAYLAY, 1997), afirma-se que uma solução para a melhoria na documentação de requisitos é a prática de observar os critérios de qualidade dos requisitos, apresentados previamente neste trabalho na seção de *Qualidade dos Requisitos*. Os métodos ágeis, apresentados anteriormente neste trabalho (capítulo 2, *Fundamentação do Processo*), apostam na intensificação da comunicação, especialmente face-a-face, na qualidade da equipe, na prototipação, na implantação rápida de partes essenciais do sistema, na refatoração e nas demais práticas ágeis como forma de melhorar a compreensão geral da equipe sobre os requisitos do sistema.

3.3 Engenharia de Requisitos no CMMI

Conforme mencionado anteriormente, os modelos do CMMI são descritos em termos de áreas de processo, metas e práticas. De acordo com (SOFTWARE ENGINEERING INSTITUTE, 2002):

> ***Áreas de processo*** *são conjuntos de* ***práticas*** *relacionadas de uma determinada área que, quando executadas coletivamente, satisfazem um conjunto de* ***metas*** *consideradas importantes para causarem uma melhoria significativa naquela área.*

Nas próximas seções, são descritas as áreas de processo relacionadas à engenharia de requisitos no CMMI: Gerência de Requisitos e Desenvolvimento de Requisitos. Para cada área de processo, são apresentadas suas metas e as práticas que devem ser executadas para que se alcancem estas metas.

3.3.1 Área de Processo: Gerência de Requisitos

Esta área de processo controla todos os requisitos do sistema, desde as necessidades do cliente até os requisitos técnicos derivados de restrições impostas por escolhas de arquitetura e até mesmo restrições no processo de desenvolvimento do sistema. A organização destes requisitos está ilustrada na figura 3.5, da seção *Requisitos do Cliente e Requisitos do Produto*, reproduzida abaixo.

Figura 3.5: Níveis de Requisitos RUP e CMMI

Também faz parte da gerência de requisitos certificar-se de que os requisitos mencionados estão coerentes entre si e em conformidade com os planos de execução do projeto.

A gerência de requisitos envolve, ainda, o controle de mudanças em requisitos, utilizando mecanismos de suporte a análises de impacto que possam ser necessárias em função das mudanças.

A tabela abaixo ilustra a meta da Gerência de Requisitos no CMMI e as práticas a ser executadas para o alcance desta meta.

Tabela 3.1: Área de Processo de Gerência de Requisitos

Área de Processo: Gerência de Requisitos (*Requirements Management*)	
Meta	**Práticas**
(SG1) Gerenciar Requisitos "Os requisitos são gerenciados e inconsistências com planos de projeto e produtos de trabalho são identificadas"	(SP1.1) Compreender os requisitos.
	(SP1.2) Obter comprometimento com os requisitos.
	(SP1.3) Gerenciar mudanças nos requisitos.
	(SP1.4) Manter rastreabilidade bidirecional dos requisitos.
	(SP1.5) Identificar inconsistências entre os resultados do projeto e os requisitos.

3.3.1.1 Meta: Gerenciar Requisitos

A gerência de requisitos, neste contexto, inclui a documentação das dependências entre requisitos, o controle de mudanças sobre estes e a identificação e correção de inconsistências entre os requisitos e os artefatos do projeto (ex: documentos, código, planos do projeto, outros requisitos).

As práticas executadas para alcançar a meta da gerência de requisitos são:

- *Compreender os Requisitos*: identificar os fornecedores de requisitos; obter aprovação destes fornecedores sobre os requisitos documentados no projeto para assegurar-se de que existe uma visão comum sobre estes; estabelecer critérios objetivos de aceitação dos requisitos.

- *Obter comprometimento com os requisitos*: certificar-se de que os participantes do projeto estão comprometidos com as atividades que precisam ser executadas para que os requisitos sejam desenvolvidos. Ou seja, certificar-se de que os planos de projeto incluem atividades correspondentes aos requisitos identificados, identificando os responsáveis por estas atividades. Esta atividade pode envolver a realização de avaliações de impacto e negociações para a acomodação de mudanças.
- *Gerenciar mudanças nos requisitos*: levando em conta o fato de que a mudança é inevitável, deve-se fornecer meios para a identificação da necessidade de mudanças, sua documentação, análise de impacto, decisão sobre a implementação ou não da mudança e ajustes decorrentes da mesma, antes de sua execução. As características da mudança e os argumentos para sua aprovação ou rejeição devem ser registrados para avaliações futuras.
- *Manter rastreabilidade bidirecional dos requisitos*: manter rastreabilidade bidirecional entre os requisitos e os artefatos do projeto (requisitos de mais baixo nível, documentos, arquivos, código, testes, planos de trabalho).
- *Identificar inconsistências entre os resultados do projeto e os requisitos*: encontrar inconsistências entre os requisitos e os artefatos do projeto, inclusive nos planos de desenvolvimento dos requisitos, e iniciar a ação corretiva para resolvê-las.

3.3.2 Área de Processo: Desenvolvimento de Requisitos

O Desenvolvimento de Requisitos compreende a investigação, análise, validação, documentação e comunicação de *requisitos do cliente* e a tradução de destes em *requisitos de produto* e de *componente de produto*. O desenvolvimento de requisitos se distribui ao longo do ciclo de vida do projeto.

A tabela abaixo resume as metas e práticas da área de processo de Desenvolvimento de Requisitos, descritas nas seções seguintes.

Tabela 3.2: Área de Processo de Desenvolvimento de Requisitos

Área de Processo: Desenvolvimento de Requisitos (*Requirements Development*)	
Met	**Práticas**
(SG1) Desenvolver requisitos do cliente “Necessidades, expectativas, restrições e interfaces dos *stakeholders* são levantadas e traduzidas em requisitos do cliente”	(SP1.1) Elicitar necessidades.
	(SP1.2) Desenvolver os requisitos do cliente.

Meta	Práticas
(SG2) Desenvolver Requisitos do Produto “Requisitos do cliente são refinados e elaborados para desenvolver requisitos de produto e de componentes de produto.”	(SP2.1) Estabelecer requisitos de produto e de componentes de produto.
	(SP2.2) Alocar requisitos de componente de produto.
	(SP2.3) Identificar requisitos de interface.
Meta	**Práticas**
(SG3) Analisar e Validar Requisitos “Os requisitos são analisados e validados, e uma definição de funcionalidade requerida é desenvolvida.”	(SP3.1) Estabelecer conceitos operacionais e cenários.
	(SP3.2) Estabelecer uma definição de funcionalidade requerida.
	(SP3.3) Analisar requisitos.
	(SP3.4) Analisar requisitos para alcançar equilíbrio.
	(SP3.5) Validar requisitos com métodos abrangentes.

3.3.2.1 Meta: Desenvolver Requisitos do Cliente

Desenvolver requisitos do cliente significa listar, definir, analisar e refinar as necessidades e restrições informadas pelos *stakeholders* do sistema. Para que esta meta seja alcançada, o CMMI recomenda as seguintes práticas:

- *Elicitar necessidades*: obter, elaborar e documentar as necessidades, expectativas e restrições dos *stakeholders* para todas as fases do ciclo de vida do sistema; refinar estas informações, analisando suas necessidades implícitas.
- *Desenvolver os requisitos do cliente*: traduzir as necessidades levantadas em requisitos do cliente; refinar ainda mais a lista de necessidades, resolvendo conflitos e completando lacunas nas definições obtidas dos *stakeholders*; incluir restrições do cliente com relação à verificação (conformidade do sistema com relação aos requisitos definidos) e validação (conformidade do sistema com relação às necessidades do usuário).

3.3.2.2 Meta: Desenvolver Requisitos do Produto

Esta meta trata da definição de requisitos mais detalhados e precisos: requisitos de produto e de componente de produto. São derivados das necessidades dos *stakeholders* e complementados com requisitos decorrentes de restrições técnicas, de escolhas de arquitetura da solução e de outras considerações semelhantes.

Estes requisitos são relacionados a funções, objetos, processos, testes e inclusive pessoas, em um processo chamado de *alocação de requisitos*. Após a execução deste processo, os requisitos são definidos como *alocados*.

As práticas que garantem o alcance da meta de desenvolvimento de requisitos do produto são:

- *Estabelecer requisitos de produto e de componentes de produto*: consiste em traduzir os requisitos do cliente em um conjunto de requisitos de produto, descritos em terminologia técnica, que podem ser utilizados, por exemplo, para decisões de projeto (*design*). Estes requisitos podem originar-se não apenas de requisitos do cliente mas também de restrições técnicas, necessidade de conformidade com padrões para conexão de sistemas, etc. Esta atividade também se preocupa em definir a relação entre os requisitos (rastreabilidade) para que possa ser realizada uma análise de impacto quando houver uma possível requisição de mudança, no futuro.
- *Alocar requisitos de componente de produto*: significa *alocar* os requisitos a funções, componentes de produto e restrições de projeto (*design*). Também inclui a tarefa de relacionar estes requisitos entre si.
- *Identificar requisitos de interface*: identificar interfaces entre funções e entre objetos, entre produtos e entre componentes de produtos. Estas interfaces passam a fazer parte da definição da arquitetura do sistema.

3.3.2.3 Meta: Analisar e Validar Requisitos

Esta meta dá suporte às duas metas anteriores, ou seja, suas práticas apóiam o desenvolvimento de requisitos do cliente, de produto e de componente de produto. Os objetivos são: a definição da funcionalidade requerida; a determinação do impacto do ambiente operacional definido sobre a habilidade de satisfazer as necessidades dos *stakeholders*; a validação dos requisitos com o intuito de aumentar a probabilidade de que o sistema resultante funcione como o cliente espera.

As práticas necessárias para alcançar esta meta são:

- *Estabelecer conceitos operacionais e cenários*: o conceito operacional define a interação entre o produto, o usuário e o ambiente, satisfazendo necessidades operacionais, de manutenção e de suporte. Um cenário é uma seqüência de eventos que podem ocorrer durante o uso de um sistema, que é utilizada para explicitar necessidades dos stakeholders. Esta prática recomenda estabelecer e manter (desenvolver, documentar, utilizar e refinar) o conceito operacional e seus cenários associados.
- *Estabelecer uma definição de funcionalidade requerida*: desenvolver, documentar, utilizar e revisar uma definição de funcionalidade requerida do sistema. A funcionalidade requerida é representada por informações que descrevam como o produto será usado (ex: ações, seqüências, entradas, saídas, etc.). Exemplos de representação desta arquitetura podem ser casos de uso detalhados ou diagramas de atividade.
- *Analisar requisitos*: com o maior detalhamento do sistema, existe uma melhor compreensão de seus produtos e componentes de produtos. Esta melhor compreensão pode levar a um refinamento dos requisitos. A prática de análise de requisitos inclui este refinamento, a análise da

rastreabilidade de requisitos para verificar se todos os requisitos identificados são necessários e suficientes ao sistema, analisar a qualidade dos requisitos (ver seção *Qualidades do Conjunto de Requisitos*) e a identificação de requisitos chave, que serão utilizados para medir o progresso do desenvolvimento do sistema.

- *Analisar requisitos para alcançar equilíbrio*: esta atividade se refere ao exercício de balanceamento que deve ser feito pelo cliente com o apoio da equipe do projeto. Ao definir suas necessidades e restrições, o cliente pode manifestar requisitos contraditórios. As dependências entre requisitos e as alternativas para resolução de seus conflitos de forma vantajosa devem ser apresentadas ao cliente para que este decida como ajustá-los. Estratégias comuns executadas nesta prática são as simulações e a prototipação. Uma avaliação de riscos nos requisitos pode ser utilizada para auxiliar esta prática.
- *Validar requisitos com métodos abrangentes*: os requisitos devem ser validados, ou seja, é preciso assegurar-se de que os requisitos documentados correspondem realmente às necessidades do cliente. Esta verificação deve ser feita através de técnicas comprovadas, como simulações e o uso de protótipos para que o cliente e os usuários do sistema verifiquem o significado dos requisitos acordados.

3.4 A Engenharia de Requisitos em RUP

O método proposto neste trabalho baseia-se fortemente nas atividades de RUP voltadas para a engenharia de requisitos. Assim, esta seção apresenta uma descrição em alto nível dos objetivos e atividades de RUP relacionados à engenharia de requisitos.

A Engenharia de Requisitos é tratada, em RUP, através da disciplina de *Requisitos*. Esta disciplina possui os seguintes objetivos (IBM RATIONAL CORPORATION, 2003):

- Estabelecer e manter um acordo com os clientes e outros *stakeholders* sobre o que o sistema deve fazer.
- Fornecer, aos desenvolvedores do sistema, um melhor entendimento dos requisitos do sistema.
- Definir os limites do sistema: o que faz parte do sistema e o que não faz.
- Fornecer uma base para o planejamento do conteúdo técnico das iterações do processo de desenvolvimento.
- Fornecer uma base para a estimativa de custo e de tempo para o desenvolvimento do sistema.
- Definir uma interface com o usuário do sistema, com foco nas necessidades e objetivos dos usuários.

Para alcançar estes objetivos, a disciplina de Requisitos de RUP segue o fluxo de trabalho ilustrado no diagrama de atividades da figura a seguir:

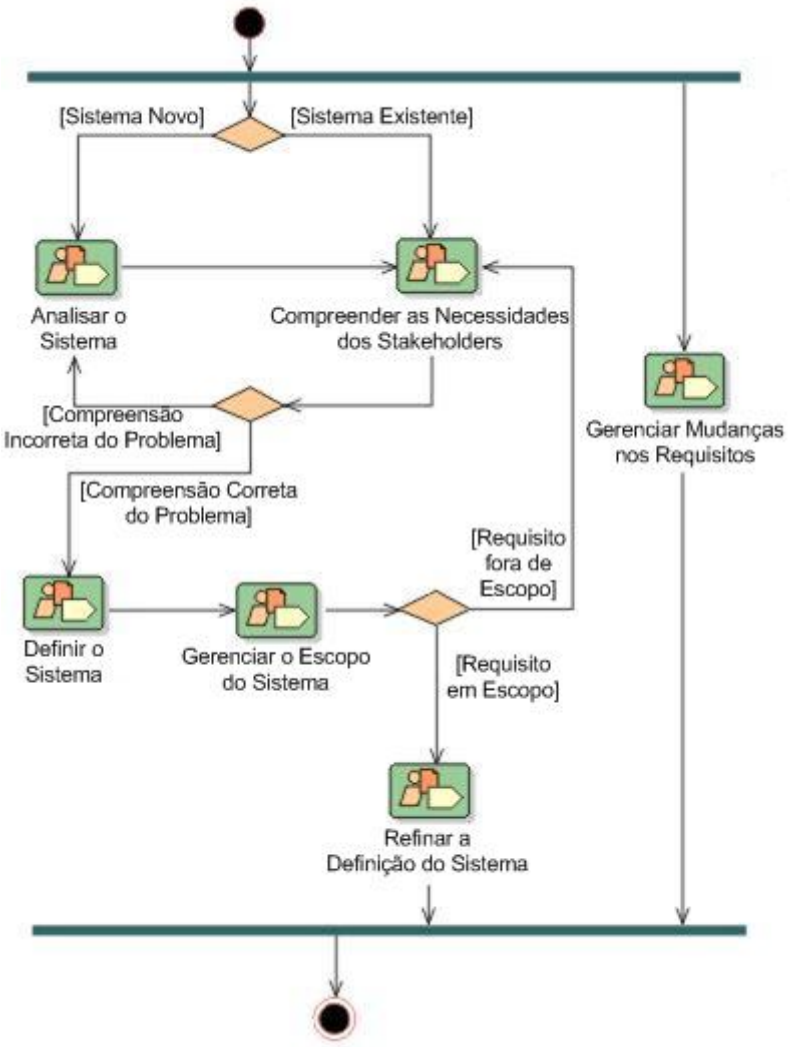


Figura 3.6: Fluxo de Trabalho de Requisitos, adaptada de (IBM RATIONAL CORPORATION, 2003)

O fluxo de trabalho apresentado na figura anterior possui os seguintes sub-fluxos:

- Analisar o problema;
- Compreender as necessidades dos *stakeholders*;
- Definir o sistema;
- Gerenciar o escopo do sistema;
- Refinar a definição do sistema;
- Gerenciar mudanças nos requisitos.

Cada um destes sub-fluxos possui objetivos específicos e contém atividades executadas para alcançar estes objetivos. Os sub-fluxos, seus objetivos e atividades são brevemente descritos a seguir. Note-se que diversas atividades são executadas repetidamente, ao longo de mais de um sub-fluxo. Isso acontece porque uma mesma atividade pode ter diferentes objetivos, dependendo do status geral do projeto. Assim, a execução de uma atividade se dá com foco diferenciado, dependendo do sub-fluxo a que pertence.

3.4.1 Analisar o Problema

3.4.1.1 Representação do sub-fluxo

Figura 3.7: Analisar o Problema, adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.1.2 Objetivos

Os objetivos do sub- fluxo *Analisar o Problema* são relacionados à obtenção de uma visão uniforme do sistema por todos os envolvidos e uma descrição clara, detalhada e concisa desta visão. Tais objetivos são:

- *Obter um acordo quanto ao problema que está sendo resolvido e suas causas*. Ou seja, certificar-se de que todas as partes interessadas concordam quanto ao problema a ser resolvido. Este objetivo também se preocupa em desenvolver uma terminologia comum para utilização ao longo da documentação do sistema.
- *Identificar os stakeholders*. Conforme descrito nas seções de definição deste trabalho, identificar os *stakeholders* é uma tarefa essencial no desenvolvimento de uma solução efetiva.
- *Definir os limites do sistema*. Os limites do sistema são a fronteira entre o que faz parte da solução e o que está ao redor da mesma, ou seja, entidades externas que interagem com o sistema.
- *Identificar restrições impostas ao sistema*. Conforme mencionado anteriormente neste trabalho, restrições limitam o grau de liberdade que se tem no desenvolvimento de uma solução. As restrições podem ser de ordem econômica, política, técnica ou ambiental e podem estar relacionadas a prazos, recursos ou orçamento.

3.4.1.3 Atividades Envolvidas

As atividades necessárias para alcançar os objetivos do sub-fluxo *Analisar o Problema* são:

- *Capturar um Vocabulário Comum*: definir um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, especialmente na descrição de casos de uso.
- *Desenvolver o Plano de Gerência de Requisitos*: desenvolver um plano para a documentação de requisitos, seus atributos e orientações para

rastreabilidade e gerência de requisitos do produto. Este plano especifica mecanismos de informação e controle que serão coletados e utilizados para medir, relatar e controlar mudanças nos requisitos do produto.

- *Encontrar Atores e Casos de Uso*: delinear a funcionalidade do sistema, utilizando a modelagem de casos de uso; definir o que será tratado pelo sistema e o que será tratado fora do mesmo; definir quem (pessoas) e o que (sistemas, dispositivos) estará interagindo com o sistema; dividir o modelo em pacotes com atores e casos de uso; criar diagramas de modelos de casos de uso; desenvolver a descrição inicial dos casos de uso do modelo.
- *Desenvolver a Visão*: obter um acordo sobre que problemas precisam ser resolvidos; identificar os *stakeholders* do sistema; definir os limites do sistema; descrever as principais características do sistema.

3.4.2 Compreender as Necessidades dos Stakeholders

3.4.2.1 Representação do sub-fluxo

Figura 3.8: Compreender Necessidades Stakeholders

adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.2.2 Objetivos

O principal objetivo deste sub-fluxo é obter informações de todos os *stakeholders* do projeto, de forma a compreender suas necessidades, para que estas sejam usadas como base para a definição das características do sistema. As características do sistema são documentadas no documento de Visão.

As necessidades dos *stakeholders* podem ser representadas por regras de negócio, requisitos de melhoria, listas de requisições. O levantamento destas informações pode ser feito de formas variadas, incluindo o recebimento de documentos produzidos pelos *stakeholders*, a realização de entrevistas e workshops de requisitos, a condução de reuniões entre os envolvidos e outras técnicas de elicitação de requisitos.

É importante observar que, embora este sub-fluxo seja executado principalmente nas fases de Iniciação e Elaboração, suas atividades também são importantes durante o decorrer do projeto, já que novas necessidades de negócio podem gerar mudanças (controladas) no escopo do sistema.

3.4.2.3 Atividades Envolvidas

As atividades necessárias para alcançar estes objetivos são:

- *Capturar um Vocabulário Comum*: continuar a definição de um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, incluindo conceitos aprendidos ao longo das demais atividades do projeto.
- *Elicitar os Requisitos dos Stakeholders*: compreender quem são os stakeholders do projeto e quais são seus pontos de vista; listar os requisitos e necessidades dos stakeholders e as conseqüentes características que o sistema deve apresentar; priorizar estes requisitos.
- *Desenvolver a Visão*: complementar e refinar as principais características do sistema, definidas no sub-fluxo *Analisar o Problema*.
- *Encontrar Atores e Casos de Uso*: continuar a procura por atores e casos de uso; refinar os modelos já criados complementando e melhorando descrições de atores e principalmente a descrição inicial dos fluxos de execução de casos de uso; ajustar interações entre estes elementos, de acordo com a compreensão atual do sistema (que fica mais elaborada ao longo de cada fase do projeto).
- *Gerenciar Dependências*: gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.

3.4.3 Definir o Sistema

3.4.3.1 Representação do sub-fluxo

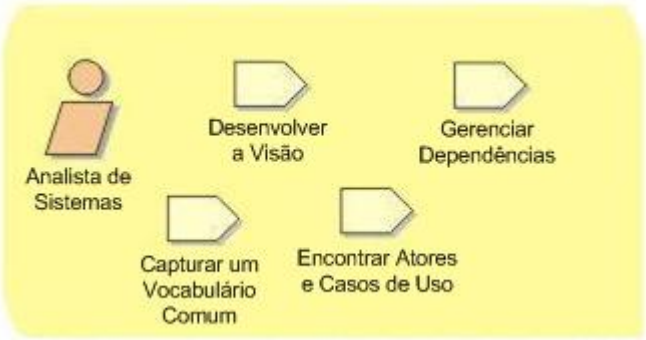


Figura 3.9: Definir o Sistema, adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.3.2 Objetivos

Os objetivos do sub-fluxo de definição do sistema são a uniformização do conhecimento e o refinamento da definição de funcionalidades que o sistema deve apresentar. Assim, as atividades deste sub-fluxo devem alinhar o conhecimento da equipe sobre o sistema em desenvolvimento; refinar os requisitos já registrados (Visão) e seus atributos; descrever os fluxos de execução de alguns casos de uso e, por fim, avaliar os resultados parciais.

3.4.3.3 Atividades Envolvidas

As atividades necessárias para alcançar os objetivos descritos acima são:

- *Desenvolver a Visão*: complementar e refinar as características do sistema documentadas até a execução deste sub-fluxo.
- *Gerenciar Dependências*: gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.
- *Capturar um Vocabulário Comum*: continuar a definição de um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, incluindo conceitos aprendidos ao longo das demais atividades do projeto.
- *Encontrar Atores e Casos de Uso*: continuar a procura por atores e casos de uso; refinar os modelos já criados complementando e melhorando descrições de atores e principalmente de casos de uso; ajustar interações entre estes elementos, de acordo com a compreensão atual do sistema (que fica mais elaborada ao longo de cada fase do projeto). O foco principal desta atividade dentro do sub-fluxo Definir o Sistema é descrever os fluxos de execução dos casos de uso.

3.4.4 Gerenciar o Escopo do Sistema

3.4.4.1 Representação do sub-fluxo

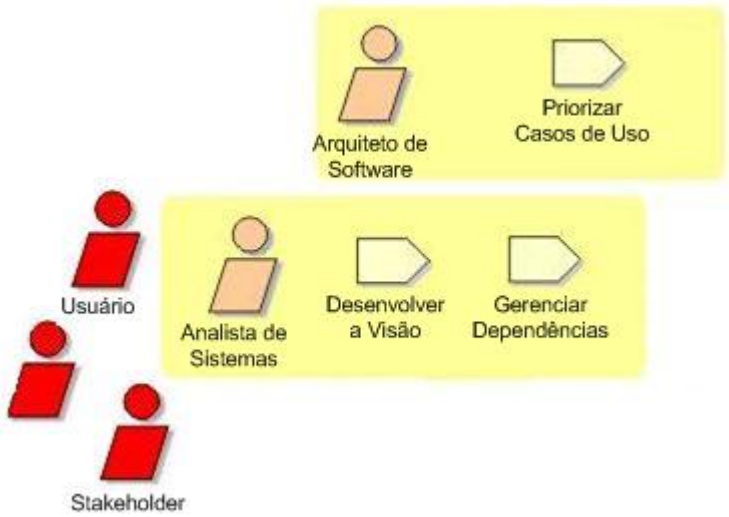


Figura 3.10: Gerenciar o Escopo do Sistema, adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.4.2 Objetivos

O sub-fluxo *Gerenciar o Escopo do Sistema* tem o objetivo de assegurar que os objetivos do projeto serão alcançados. As atividades executadas levam à seleção dos requisitos do sistema a ser desenvolvidos em cada iteração. Estes requisitos são escolhidos de forma a balancear as variáveis de escopo e recursos (tempo, pessoal, orçamento, etc.).
Os requisitos selecionados para o projeto (ou para uma iteração específica) representam seu escopo. A seleção é feita com base nos atributos dos

requisitos, que incluem: importância e impacto para o negócio; complexidade; impacto na arquitetura; estabilidade; risco; etc.
A definição e o refinamento dos atributos de requisitos, bem como a compreensão da rastreabilidade entre estes requisitos, também são objetivos deste sub-fluxo.

3.4.4.3 Atividades Envolvidas

As atividades necessárias para alcançar os objetivos deste sub-fluxo são:

- *Priorizar Casos de Uso*: definir os casos de uso e cenários que serão analisados em cada iteração; definir os principais casos de uso e cenários que são significativos do ponto de vista das funcionalidades do sistema, do ponto de vista da arquitetura e em contextos específicos como performance, adaptabilidade ou outro ponto importante em termos de restrições do sistema.
- *Desenvolver a Visão*: complementar e refinar as características do sistema documentadas até a execução deste sub-fluxo. É importante destacar que mudanças feitas na documentação de Visão fora da etapa de Iniciação devem ser tratadas como requisições de mudança, ou seja, devem seguir o processo de verificação de impacto, incluindo atividades como *Gerenciar Dependências* e *Gerenciar Requisições de Mudança*.
- *Gerenciar Dependências*: gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.

3.4.5 Refinar a Definição do Sistema

3.4.5.1 Representação do sub-fluxo

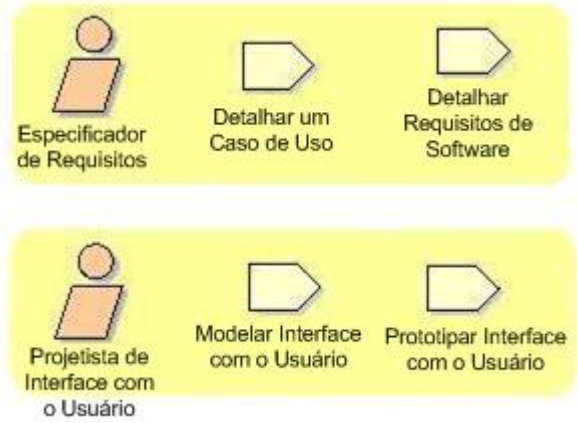


Figura 3.11: Refinar a Definição do Sistema, adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.5.2 Objetivos

Os objetivos deste fluxo de trabalho levam ao refinamento da solução descrita para o sistema. Neste sub-fluxo, os requisitos levantados inicialmente são detalhados o suficiente para que seu projeto possa ser iniciado. Estes objetivos são:

- Detalhar o fluxo de execução dos casos de uso;
- Detalhar especificações suplementares, referentes a requisitos que não se encaixam nas descrições de casos de uso;
- Desenvolver especificações de requisitos de software, quando necessário;
- Modelar e prototipar a interface de usuário do sistema.

3.4.5.3 Atividades Envolvidas

As atividades necessárias para alcançar estes objetivos são:

- *Detalhar um Caso de Uso*: descrever detalhadamente um caso de uso que já tenha sido identificado em atividades anteriores. O caso de uso também já deve ter sido brevemente descrito e seus principais passos identificados. A descrição detalhada inclui uma explicação minuciosa do fluxo de execução (principais passos e iterações), pré e pós-condições, fluxos de erro e fluxos alternativos e demais informações relacionadas ao caso de uso. Esta descrição é utilizada para verificar que a equipe e o cliente compreendem o funcionamento desejado do sistema da mesma forma.
- *Detalhar os Requisitos de Software*: recolher, detalhar e organizar o conjunto de artefatos que descreve, de forma completa, os requisitos de software do sistema. Os principais artefatos são: os atributos dos requisitos, as especificações suplementares (restrições que não se aplicam ao contexto específico de um caso de uso) e a especificação de requisitos de software, listando todos os requisitos do sistema (geralmente uma lista de casos de uso).
- *Modelar a Interface com o Usuário*: construir um modelo de interface do sistema com o usuário, considerando os atores, os fluxos de eventos dos casos de uso e requisitos de usabilidade; identificar *classes de interface* (*boundary classes*) entre o sistema e seus usuários; descrever a interação entre essas classes e complementar os demais modelos do sistema a partir das novas definições feitas.
- *Prototipar a Interface com o Usuário*: criar um protótipo de interface do sistema com o usuário, partindo do projeto de elementos da interface (ex: janelas, páginas web) até sua implementação; avaliar os resultados com os usuários.

3.4.6 Gerenciar Requisições de Mudança

3.4.6.1 Representação do sub-fluxo

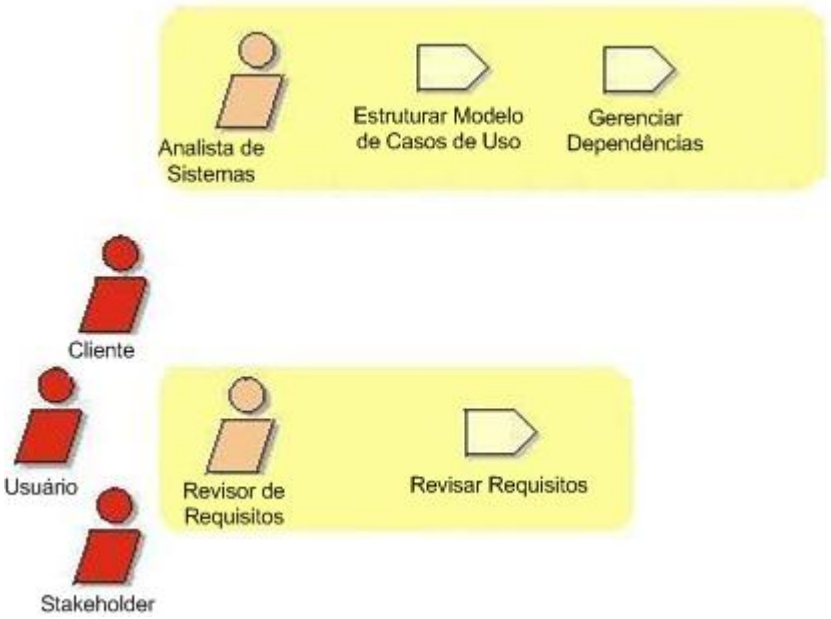


Figura 3.12: Gerenciar Requisições de Mudança, adaptada de (IBM RATIONAL CORPORATION, 2003)

3.4.6.2 Objetivos

Ao longo do projeto, mudanças nas necessidades dos stakeholders são esperadas. Tais mudanças ocorrem em vista de novas necessidades de negócio, melhor compreensão do sistema por parte de todos os envolvidos, problemas e soluções encontrados nas primeiras iterações do desenvolvimento de *software*.
A gerências destas mudanças tem os seguintes objetivos:

- Avaliar requisições de mudança submetidas formalmente e determinar seu impacto no conjunto de requisitos existente. O histórico das alterações de um requisito é uma parte de grande importância nesta avaliação. Assim, as decisões tomadas devem ser devidamente registradas, assim como os argumentos utilizados na tomada de decisão.
- Estruturar o modelo de casos de uso, atualizando-o de acordo com as mudanças aceitas.
- Definir atributos e rastreabilidade entre requisitos de forma apropriada ao escopo corrente.
- Verificar formalmente que os resultados da disciplina de Requisitos estão de acordo com a visão que o cliente tem do sistema.

3.4.6.3 Atividades Envolvidas

As atividades necessárias para alcançar os objetivos do sub-fluxo *Gerenciar Requisições de Mudança* são:

- *Estruturar o Modelo de Casos de Uso*: refinar o modelo de casos de uso, extraindo comportamento comum para casos de uso abstratos e

definindo relações “*include*” ou “*extend*”; procurar novos atores abstratos que definam papéis compartilhados por vários atores.

- *Gerenciar Dependências*: gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.
- *Revisar os Requisitos*: verificar formalmente que os resultados da disciplina de Requisitos estão de acordo com as expectativas do cliente; revisar o estado atual dos requisitos e as requisições de mudança.

3.4.7 Distribuição das Atividades por Fase de Desenvolvimento

As atividades da disciplina de Requisitos de RUP são distribuídas ao longo das fases de Iniciação, Elaboração, Construção e Transição de acordo com o planejamento do projeto e de cada uma de suas iterações. Contudo, o fluxo de execução da disciplina de Requisitos e as características específicas de cada atividade levam a um certo posicionamento da atividade dentro das fases de RUP.

O processo de engenharia de requisitos proposto neste trabalho, apresentado mais adiante, utiliza esta tendência de posicionamento de atividades para simplificar os sub-fluxos utilizados. Tal processo define a maioria de suas atividades de forma a serem executadas em apenas um sub-fluxo. Isso faz com que o foco da atividade fique mais claro, já que esta não é executada em pontos variados do processo, facilitando sua compreensão pela equipe de trabalho.

3.5 Métodos Ágeis e a Engenharia de Requisitos

Embora os métodos ágeis sejam caracterizados pelo foco em princípios, valores e práticas, e não em uma definição rigorosa de processos, estes métodos oferecem diversas orientações sobre como executar atividades da engenharia de requisitos.

O elemento básico da engenharia de requisitos em diversos métodos ágeis é a estória do usuário, definida como parte do método de XP. Uma estória do usuário é uma funcionalidade, um requisito de alto nível do sistema, detalhado apenas o suficiente para permitir que os desenvolvedores possam estimar o esforço necessário para implementá-la. Em (JEFFRIES, 2001), são descritos três aspectos básicos das estórias do usuário:

1. *Cartões*: são utilizados como meio de armazenamento das estórias do usuário. O objetivo dos cartões é registrar a estória com o mínimo de informações que permitam identificá-la durante o processo de desenvolvimento. Inicialmente, o cartão pode conter apenas o nome da estória. Em sessões de planejamento, podem ser registrados sua prioridade e seu custo estimado. O cartão será entregue ao programados no momento da implementação.
2. *Conversação*: o conteúdo da estória é definido através da conversação, ao longo do tempo, geralmente durante o planejamento, ao estimar de esforço e ao planejar a implementação. Embora a descrição do requisito

seja feita verbalmente, os detalhes podem ser registrados em documentos, *storyboards*, planilhas, etc.

3. *Confirmação*: é alcançada através da realização dos testes de aceitação, que são criados pelo cliente e podem ser implementados pela equipe de desenvolvimento. Estes testes contêm quesitos a serem verificados para que se considere a estória do usuário implementada.

Em (AMBLER, 2005), é apresentado um conjunto de melhores práticas ágeis para a engenharia de requisitos. Algumas destas práticas são apresentadas abaixo:

- Os *stakeholders* participam ativamente, levantando, definindo e priorizando os requisitos, e são inclusive convidados a modelar e documentar os requisitos juntamente com os desenvolvedores. Para que esta prática seja facilitada, é importante ensinar, aos *stakeholders*, as técnicas utilizadas na engenharia de requisitos e favorecer a utilização de recursos simples, preferencialmente lápis, papel e quadros, ao invés de ferramentas que possam excluir a participação dos *stakeholders*.
- A abordagem ideal para modelagem é iniciar com uma noção geral do sistema e depois partir para o detalhamento dos requisitos.
- A documentação dos requisitos deve seguir o princípio da simplicidade, gerando apenas os documentos necessários, com o mínimo conteúdo que seja suficiente para as demais atividades do processo de desenvolvimento. O objetivo é *implementar* requisitos, não documentá-los.
- O apoio da gerência interna e externa, ou seja, da própria organização e da equipe do cliente são vitais para a aplicação de práticas ágeis de forma bem sucedida.

Em (SCHWABER, 2002), são descritas estratégias de engenharia de requisitos em processos ágeis. A colaboração entre *stakeholders* e desenvolvedores é mencionada novamente e o desenvolvimento iterativo é citado como forma de assegurar o sucesso do projeto, implementando partes do sistema para validação pelo cliente. Também é recomendada a realização de reuniões diárias, para disseminar o status do projeto e orientar as atividades da equipe, que se auto-organiza da forma que acredita ser a mais efetiva. Por fim, afirma-se que a arquitetura, a estrutura da equipe e os requisitos devem *emergir* ao longo do projeto. Para projetos maiores, contudo, a arquitetura do sistema é detalhada logo no início do projeto, simplificando a coordenação de múltiplos times.

4 PROCESSO DE ENGENHARIA DE REQUISITOS PROPOSTO

Este capítulo descreve o processo de engenharia de requisitos proposto neste trabalho. Primeiramente, é descrita a metodologia utilizada para definir o processo, sua inserção no contexto de uma organização específica e a fundamentação teórica do processo, baseada em RUP e CMMI. Após isso, são apresentados os componentes básicos do processo, seguidos de uma descrição detalhada dos fluxos de atividades do processo. Por fim, são apresentadas orientações sobre a inserção de práticas de métodos ágeis nas atividades do processo.

4.1 Metodologia de Concepção do Processo Proposto

4.1.1 Contextualização do Processo

O processo de engenharia de requisitos proposto neste trabalho foi concebido e implantado em uma organização de desenvolvimento de sistemas *software*. Este processo se insere em um processo global de desenvolvimento elaborado pela organização com o objetivo de obter melhorias significativas em produtividade e qualidade de seus produtos de *software*.

O processo global de desenvolvimento segue o ciclo de vida de RUP, com as fases de Iniciação, Elaboração, Construção e Transição. Para cada fase, é definido um conjunto de fluxos de atividades, que representam a descrição do processo. Esta documentação é construída sob a forma de páginas navegáveis, acessadas por navegador *web*, disponíveis na *intranet* da organização. A figura abaixo mostra a página que documenta os fluxos de atividade da fase de Iniciação, com a opção de navegação para o detalhamento das atividades do fluxo *Definir o Escopo do Sistema*.

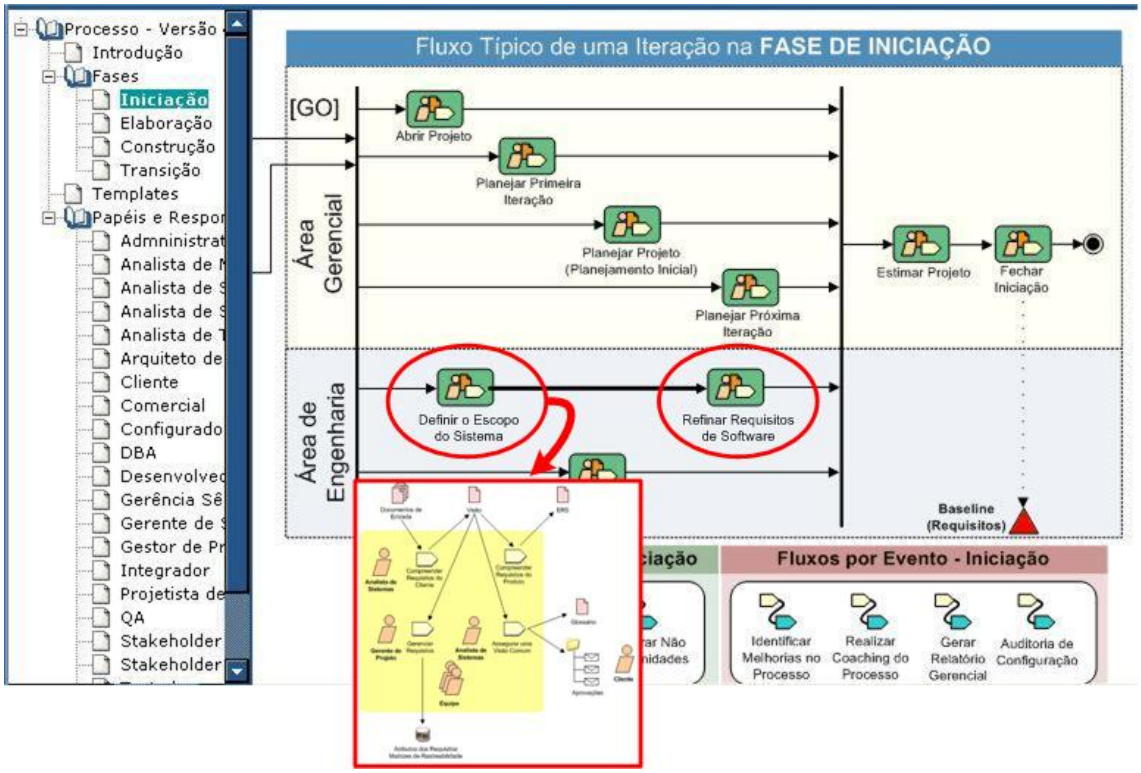


Figura 4.1: Fase de Iniciação do processo de desenvolvimento global

Todos os aspectos relacionados à engenharia de requisitos dentro do processo global de desenvolvimento estão descritos neste trabalho.

4.1.2 Objetivos e Características Fundamentais

Conforme descrito no capítulo 1, *Introdução*, o processo de engenharia de requisitos proposto neste trabalho baseou-se nos enfoques CMMI, RUP e métodos ágeis.

Com o objetivo de alcançar qualidade e melhoria contínua, o processo busca conformidade com as áreas de processo do CMMI de Gerência de Requisitos (nível 2) e Desenvolvimento de Requisitos (nível 3). Outro ganho obtido como conseqüência da conformidade com CMMI é o reconhecimento nacional e internacional da maturidade da organização.

Os componentes do processo (atividades, papéis, produtos de trabalho) são baseados no *framework* de processo RUP que, além de ser uma metodologia reconhecida e destacada no mercado e na academia, já é utilizado informalmente nas atividades das equipes de desenvolvimento da organização.

Práticas de métodos ágeis são recomendadas para utilização durante a execução do processo, com o intuito de aumentar produtividade e qualidade.

Mais um quesito que orientou a elaboração do processo foi a definição do seguinte princípio básico: *o processo deve primar pela praticidade e eficiência, evitando atividades burocráticas e geração de artefatos que não ofereçam índices interessantes de retorno do investimento*.

Conhecimentos adquiridos durante atividades de consultoria a uma empresa cliente, cujo processo adotado obteve certificação no nível 2 de maturidade do CMMI, também foram aproveitados na concepção do processo proposto.

4.1.3 Evolução do Processo Concebido

O primeiro passo na concepção do processo foi a uniformização dos conceitos relacionados à engenharia de requisitos utilizados pelas abordagens fundamentais utilizadas. Este passo resultou nas definições apresentadas no capítulo 3, *Processos e Práticas da Engenharia de Requisitos*, sobretudo na seção *Requisitos: Conceitos e Definições*.

Os próximos passos, executados de forma iterativa, foram: pesquisa, definição, implantação, avaliação e ajuste. A primeira iteração envolveu a pesquisa e estudo abrangente das abordagens fundamentais do processo e a definição de sua versão inicial, compatível com a área de processo de Gerência de Requisitos, pertencente ao nível de maturidade 2 do CMMI. O processo de institucionalização foi iniciado e os projetos piloto foram definidos e iniciados. Após isso, os resultados parciais da adoção do processo foram analisados.

A próxima iteração de concepção do processo foi caracterizada por pesquisas mais direcionadas, com o estudo de assuntos específicos para a evolução do processo. Foram incluídos atividades e artefatos com o objetivo de adquirir conformidade com a área de processo de Desenvolvimento de Requisitos, do nível de maturidade 3 do CMMI. Também foram adicionadas ao processo orientações para a utilização de princípios, valores e práticas dos métodos ágeis.

Na segunda iteração, embora tenha sido adotada uma área de processo do nível 3 de maturidade do CMMI, isso aconteceu apenas no processo específico de engenharia de requisitos, e não no processo global da organização. Este fato resultou em problemas de institucionalização, conforme será descrito mais adiante neste trabalho.

4.2 Elementos Básicos

Os elementos básicos de formação do processo proposto são os utilizados por RUP, para descrever seus aspectos estáticos: papéis, artefatos, atividades e fluxos de atividades (ver capítulo 2, seção *O Processo Unificado da Rational*).

Os três fluxos de atividades de engenharia de requisitos são:

1. Definir o Escopo do Sistema;
2. Refinar Requisitos de Software;
3. Gerenciar Mudanças.

Cada um desses fluxos é composto por um conjunto de atividades, que podem ser ordenadas ou não. Cada atividade é exercida por um membro da equipe a desempenhar um certo papel. A execução de uma atividade pode utilizar artefatos como entrada e pode gerar artefatos de saída. Algumas atividades aparecem em mais de um fluxo de atividades, e foram denominadas *atividades recorrentes*.

As seções a seguir descrevem os papéis e artefatos relacionados a atividades do processo de engenharia de requisitos proposto neste trabalho.

4.2.1 Papéis

O conceito de *papel* na equipe de desenvolvimento é o mesmo utilizado em RUP, ou seja, a descrição do comportamento e responsabilidades de um

indivíduo ou grupo de indivíduos trabalhando juntos. O comportamento é descrito por meio das atividades associadas àquele papel e as responsabilidades são definidas com relação aos artefatos criados, modificados ou controlados por ele.

Os papéis definidos para o processo de engenharia de requisitos proposto neste trabalho são:

- Equipe: toda a equipe de desenvolvimento;
- *Analista de Sistemas*: o principal responsável pelas atividades de engenharia de requisitos: elicitação, análise, especificação, validação e gerência de requisitos.
- *Gerente de Projeto*: responsável pelo bom andamento do projeto, monitoração de riscos, alocação de recursos e demais atividades de planejamento.
- *Cliente*: representa um *stakeholder* do sistema, pertencente à organização cliente e que possui autorização para atuar como fornecedor de requisitos ao projeto.
- Comitê de Controle de Mudanças: é formado por representantes de todos os *stakeholders*. Uma configuração típica seria: o gerente de projeto, o gerente do projeto na organização cliente, um analista e um líder técnico. Em projetos pequenos, pode ser formado apenas pelo gerente de projeto.
- *Qualquer Papel*: qualquer membro da equipe de desenvolvimento ou stakeholder da organização cliente.

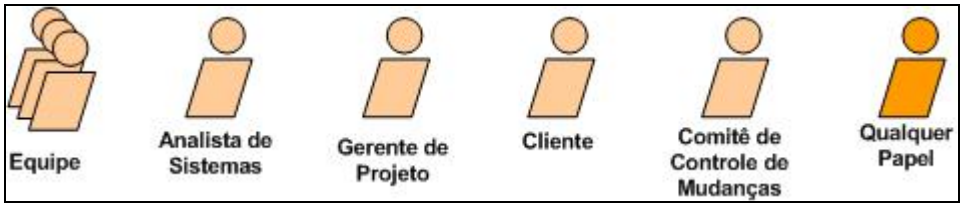


Figura 4.2: Papéis do Processo Proposto

4.2.2 Artefatos

Os artefatos produzidos durante a execução das atividades do projeto são:

- *Documentos de Entrada*: quaisquer documentos relacionados ao projeto que foram produzidos antes da fase de Iniciação. Exemplos típicos de documentos de entrada são RFPs (*Request for Proposal*s) enviados pelo cliente, a *Proposta Comercial* feita pelo departamento de vendas e aceita pelo cliente e a *Proposta Técnica*, que sugere uma versão inicial das características e módulos do sistema.
- *Atributos dos Requisitos*: são dados recolhidos ao longo do processo de análise de requisitos, desde a especificação até a aprovação da especificação final ou de posteriores mudanças. Há vários tipos de atributos de requisitos: identificadores e informações cadastrais, especificações e detalhamento, status e índices como importância para o negócio, relevância para arquitetura, estimativa de tamanho (ou complexidade). Um atributo especial é a *prioridade do requisito*, definida com base nos índices de outros atributos, e cuja definição depende de

negociações com o cliente que envolvem atividades de gerência de projeto.

- *Matrizes de Rastreabilitade*: documentam as relações de dependência entre requisitos. A rastreabilidade documentada é vertical (obrigatória) e horizontal (opcional). É possível documentar as matrizes de rastreabilidade explicitamente, através de tabelas, planilhas ou ferramentas de gerência de requisitos, ou implicitamente, através dos demais artefatos do projeto. Um exemplo de documentação implícita é a matriz que relaciona requisitos aos recursos de projeto como tempo e pessoas. Os cronogramas do projeto possuem atividades agrupadas em blocos com os mesmos nomes dos requisitos, o que resulta em uma relação implícita com os requisitos armazenados no repositório.
- *Glossário*: Registro o vocabulário comum do projeto, na linguagem do cliente. É criado no início do projeto e evolui ao longo do desenvolvimento do sistema, servindo como referência para todos os envolvidos.
- *Documento de Visão*: Neste documento é definida a visão que os envolvidos têm do produto a ser entregue, em termos das principais necessidades e características. Também registra a motivação para o desenvolvimento do sistema, através da breve descrição do principal problema de negócio a ser resolvido pelo sistema, além de descrever os limites do produto e as restrições impostas à solução. Por conter uma descrição em alto nível dos requisitos centrais pretendidos, incluindo restrições de design, serve como base contratual para requisitos técnicos mais detalhados.
- *Especificação de Requisitos de Software (ERS)*: captura todos os requisitos de software do sistema e consiste em um pacote contendo especificações funcionais e não funcionais aplicáveis. As especificações funcionais possuem dois níveis de detalhamento: uma visão geral da funcionalidade é descrita neste documento e o detalhamento minucioso de cada requisito é documentado em um documento chamado Especificação Funcional de Requisito. Os requisitos mais genéricos, alguns requisitos não funcionais e os requisitos que não podem ser bem representados nas especificações funcionais também são registrados no ERS, na seção de Especificações Suplementares.
- *Especificação Funcional de Requisitos*: complementa a Especificação de Requisitos de Software (ERS) do sistema, detalhando uma funcionalidade específica. Este documento descreve minuciosamente a interação que ocorre entre os atores (usuários ou sistemas externos) e o sistema durante a execução deste(s) requisito(s). O principal objetivo é especificar o funcionamento de uma porção do sistema de forma clara o suficiente para que: o cliente possa validar o que será desenvolvido e a equipe de desenvolvimento compreenda o que deve entregar.
- *Modelo de Domínio*: modelo de objetos inicial do sistema ou outra representação das entidades essenciais do sistema.
- *Protótipo de Interface*: descrição de uma ou mais interfaces com o usuário. Pode ser criada em vários formatos, tais como protótipos funcionais do sistema, protótipos estáticos, exemplos de tela, desenhos à mão livre capturados em meio eletrônico, etc.

- *Solicitação de Mudança*: utilizada sempre que uma mudança no projeto é requisitada. Este artefato pode ser criado por qualquer membro da equipe de desenvolvimento ou da equipe do cliente, e deve ser aprovado pelo Comitê de Controle de Mudanças. A Solicitação de Mudança é utilizada para documentar a necessidade de mudança (defeito, melhoria ou novos requisitos), informações sobre impacto e conseqüências de sua adoção e status da mudança, além das justificativas para as decisões tomadas sobre a implementação da mudança.
- *Repositório do Projeto*: contém todos os artefatos utilizados no processo de desenvolvimento do software. Pode ser um conjunto de arquivos, um banco de dados ou qualquer outra base controlada, por exemplo, por uma ferramenta de gerência de requisitos.
- *Aprovações*: comunicações oficiais enviadas pelo cliente, atestando aceitação de algum artefato produzido. Geralmente, são e-mails com a aprovação de Especificações Funcionais de Requisitos.

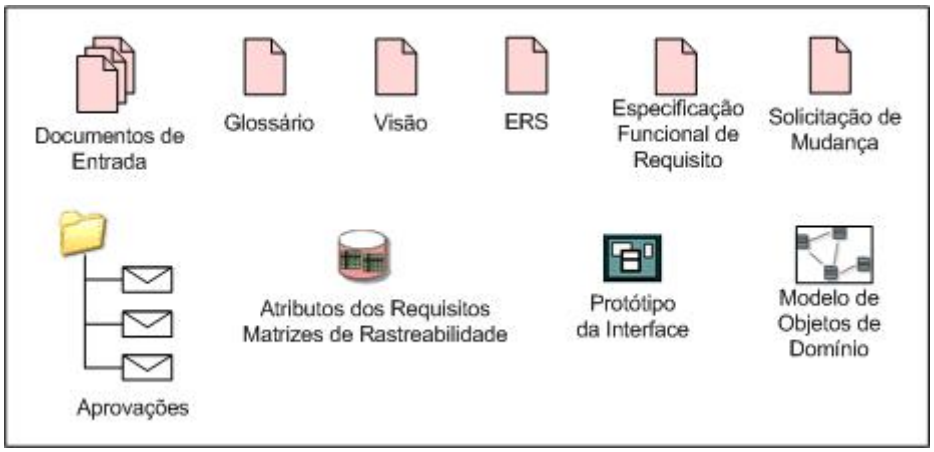


Figura 4.3: Artefatos do Processo Proposto

4.3 Atividades Recorrentes

Algumas atividades são executadas repetidamente ao longo dos fluxos de trabalho do processo de engenharia de requisitos proposto. Estas atividades dizem respeito a necessidades de controle de requisitos e de compartilhamento e reiteração do conhecimento para todos os membros da equipe de desenvolvimento. São elas:

- *Gerenciar Requisitos*: está intimamente ligada aos conceitos de gerência de requisitos, de gerência de escopo e de controle de mudanças, apresentados previamente neste trabalho.
- *Assegurar uma Visão Comum*: relaciona-se à necessidade de validação dos requisitos com os clientes e à difusão e consolidação do conhecimento sobre o sistema a todos os membros da equipe de desenvolvimento.

Estas atividades são descritas em detalhes nas seções seguintes.

4.3.1 Gerenciar Requisitos

4.3.1.1 Objetivos, Papéis e Artefatos

Esta atividade trata do controle, rastreabilidade, histórico e atributos de todos os requisitos do sistema, em todos os níveis de abstração, ou seja, necessidades, características, requisitos de software funcionais e não funcionais. Seu principal objetivo é fornecer subsídios para a gerência de escopo e o controle de mudanças no projeto.

Todos os fluxos de trabalho relacionados a requisitos influenciam os atributos de requisitos. Assim, esta atividade está presente em todos os fluxos de trabalho do processo de engenharia de requisitos proposto.

Os artefatos de entrada para esta atividade são todos aqueles relacionados a requisitos. Cada vez que um novo artefato deste tipo é criado ou modificado, esta atividade é executada, atualizando os atributos dos requisitos.e as matrizes de rastreabilidade quando necessário.

Todos os membros da equipe são responsáveis pela gerência de requisitos, mas o analista de sistemas é responsável por revisar os atributos e matrizes de rastreabilidade de requisitos freqüentemente (ao menos uma vez por iteração), para certificar-se de que o conjunto de requisitos do sistema está propriamente controlado.

A atividade *Gerenciar Requisitos* é executada em três passos:

4. Manter atributos dos requisitos.
5. Manter rastreabilidade.
6. Priorizar requisitos.

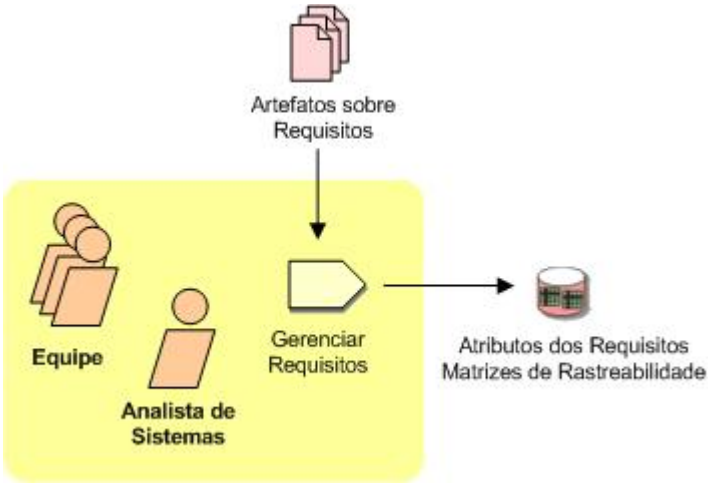


Figura 4.4: Gerenciar Requisitos

4.3.1.2 Manter Atributos dos Requisitos

Sempre que alguma atividade cria ou modifica um artefato relacionado a requisitos do sistema, seus atributos são atualizados. Os atributos de requisitos utilizados neste processo estão descritos na tabela abaixo. Os atributos obrigatórios são fundamentais para a gerência de requisitos e precisam ser sempre registrados; os atributos opcionais são sugeridos para facilitar outras atividades do processo e a decisão de adotá-los ou não faz parte das atividades de planejamento do projeto.

Tabela 4.1: Atributos de Requisitos do Processo Proposto

Atributo	**Descrição**	**Utilização**
Nome	Nome do requisito, atribuído pelo analista de sistemas no momento de sua identificação. Pode ser refinado ao longo do período de análise.	Obrigatória
Tipo	Nível de abstração do requisito: • Necessidade; • Característica; • Requisito Funcional ou • Requisito Não Funcional.	Obrigatória
Identificador	Um identificador único, que permite manter referências ao requisito, já que o nome pode mudar. A sugestão de identificador dada por este processo é: • Necessidades: NECn • Características: CARCn • Requisitos: RFn, RNFn Onde “n” é um número seqüencial, iniciado em “1”. Note-se que a seqüência de cada tipo de requisito pode ter intervalos de falha, quando requisitos descobertos inicialmente foram removidos do escopo do projeto.	Obrigatória
Tamanho	Uma medida de tamanho de requisitos. Neste processo, recomenda-se utilizar Pontos por Caso de Uso (*Use Case Points*) (SMITH, 1999).	Obrigatória
Relevância para o negócio	Este atributo destina-se a auxiliar na priorização dos requisitos. É definido pelo cliente, que deve informar ao analista a importância de um requisito para a solução de suas necessidades.	Opcional
Relevância para a arquitetura	Este atributo destina-se a auxiliar na priorização dos requisitos. É definido pelo arquiteto do sistema, que deve analisar o grau de influência que o projeto (*design*) de cada requisitos tem sobre a arquitetura geral do sistema e sobre as decisões técnicas a serem tomadas ao longo do projeto.	Opcional

Atributo	Descrição	Utilização
Prioridade	Este atributo define a ordem de desenvolvimento dos requisitos do projeto. A prioridade é utilizada para auxiliar no planejamento, na definição de cronogramas, na composição de *releases* do sistema e atividades de teste, entre outras.	Obrigatória
Status	Status atual do requisito, variando entre os seguintes: • Identificado (estado inicial); • Em desenvolvimento; • Em aprovação; • Aprovado; • Em alteração.	Obrigatória
Data de criação	Data de identificação do requisito.	Opcional
Responsável pela criação	Membro da equipe responsável pela identificação do caso de uso.	Opcional
Data da última atualização	Data da última atualização da especificação do requisito.	Opcional
Responsável pela última atualização	Membro da equipe responsável pela última atualização da especificação do requisito.	Opcional
Data de aprovação	Data em que o cliente aprovou formalmente a especificação detalhada do requisito.	Obrigatória
Responsável pela aprovação	Nome do membro da equipe do cliente que aprovou formalmente a especificação detalhada do requisito.	Obrigatória
Histórico de modificações	Diferentes versões da especificação detalhada do requisito.	Opcional

Grande parte desta tarefa é, em geral, automática, em conseqüência da utilização de ferramentas de gerência de requisitos.

4.3.1.3 Manter Rastreabilidade

Este passo da gerência de requisitos trata da criação e manutenção das matrizes de rastreabilidade dos requisitos. Conforme comentou-se anteriormente neste trabalho, a documentação das matrizes pode ser explícita ou implícita. A tabela abaixo descreve as matrizes mantidas no processo proposto, que resultam na documentação das dependências ilustradas na figura a seguir.

Tabela 4.2: Matrizes de Rastreabilidade do Processo Proposto

Matriz	Utilização
Necessidades x Características	Obrigatória
Características x Características	Opcional

Matriz	Utilização
Características x Requisitos Funcionais	Obrigatória
Características x Requisitos Não Funcionais	Obrigatória
Requisitos Funcionais x Requisitos Não Funcionais	Obrigatória
Requisitos Funcionais x Requisitos Funcionais	Opcional
Requisitos Funcionais x Especificações Funcionais	Obrigatória
Requisitos x Especificações Técnicas	Obrigatória
Requisitos x Casos de Teste	Obrigatória
Requisitos x Código	Obrigatória
Requisitos x Planos de Projeto	Obrigatória

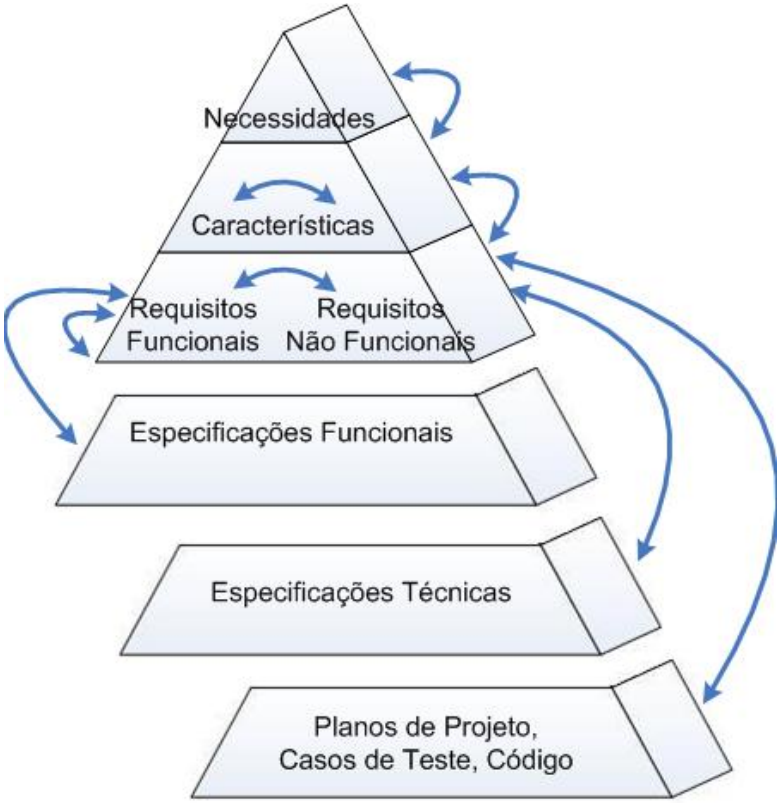


Figura 4.5: Rastrabilidade de Requisitos do Processo Proposto

Assim como a manutenção de atributos, a manutenção das matrizes de rastreabilidade ocorre ao longo de todo o processo de desenvolvimento. Após a conclusão de cada artefato do sistema que descreva os requisitos da figura acima, este artefato deve ser submetido ao controle de rastreabilidade, o que significa atualizar as matrizes relacionadas a ele.

4.3.1.4 Priorizar Requisitos

Este terceiro passo da atividade *Gerenciar Requisitos* completa o primeiro passo, de manutenção de requisitos, preenchendo o valor do atributo de *prioridade*.

As prioridades dos requisitos do sistema orientam o planejamento e a gerência de escopo. Como o processo proposto segue o ciclo de vida iterativo

e incremental, cada iteração resulta em novos conjuntos de requisitos analisados, projetados e codificados. A prioridade auxilia na determinação dos casos de uso que compõem estes conjuntos.

Dependendo do projeto, a priorização pode ter uma granularidade menor. Por exemplo, se os requisitos de um projeto são bastante complexos ou interdependentes, e se estão documentados por meio de casos de uso, pode ser registrada a priorização de *cenários* de casos de uso, de forma mais detalhada do que apenas a priorização dos casos de uso.

Os fatores que devem ser observados para determinar a prioridade dos requisitos são:

- Relevância do requisito para o negócio, definida pelo cliente e o analista de sistemas;
- Relevância do requisito para a arquitetura, definida pelo arquiteto de software;
- Riscos de projeto a serem mitigados, identificados durante atividades de planejamento e gerência do projeto;
- Dependências entre requisitos;
- Outros objetivos táticos ou restrições como por exemplo questões técnicas ainda não tratadas ou versões intermediárias do sistema a serem liberadas para validação pelo usuário.

Como o processo proposto segue as orientações de RUP quanto à divisão nas fases de Iniciação, Elaboração, Construção e Transição, a relevância para a arquitetura tende a receber maior atenção, uma vez que a arquitetura deve estar claramente definida ao final da Elaboração.

4.3.2 Assegurar uma Visão Comum

4.3.2.1 Objetivos, Papéis e Artefatos

O principal objetivo desta atividade é garantir que todos os envolvidos no projeto possuem uma visão comum do sistema em desenvolvimento. A equipe de trabalho precisa de uma visão única para que os componentes do sistema possam ser integrados propriamente e de forma a fornecerem as funcionalidades desejadas. A visão também deve ser a mesma entre a equipe de trabalho e o cliente, para que o sistema entregue possa ser utilizado da forma esperada.

A visão comum sobre o sistema deve ser construída ao longo de todo o projeto, o que faz com que esta atividade apareça em todos os fluxos de trabalho do processo de engenharia de requisitos proposto.

Os artefatos de entrada para esta atividade são todos aqueles relacionados a requisitos. Esta atividade é executada em paralelo ao desenvolvimento dos artefatos do projeto relacionados a requisitos. Por isso, todos os membros da equipe estão envolvidos, assim como os fornecedores de requisitos do sistema, ou seja, os clientes. Embora todos estes participantes estejam envolvidos, é tarefa do gerente de projeto certificar-se do bom andamento desta atividade.

Os passos utilizados para assegurar a visão comum do sistema são:

1. Realizar análise de requisitos em grupos;
2. Revisar artefatos;
3. Obter aprovações;

4. Manter Glossário.

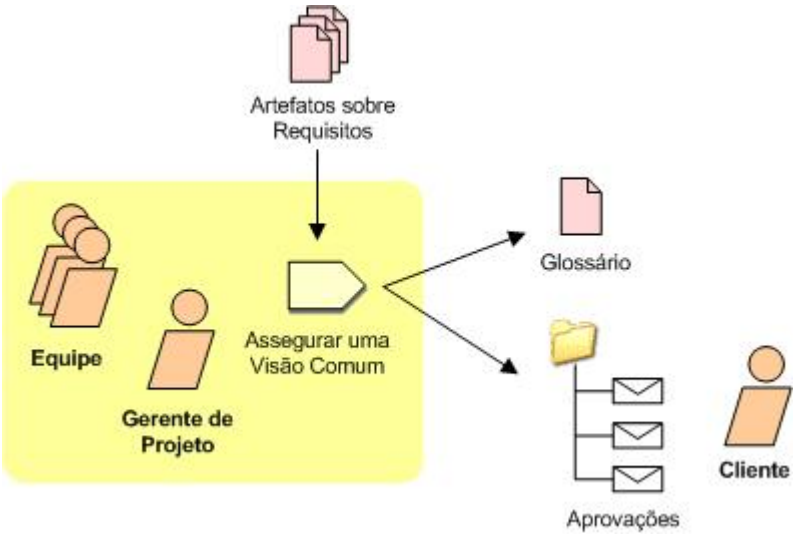


Figura 4.6: Assegurar uma Visão Comum

4.3.2.2 Realizar análise de requisitos em grupos

A análise de requisitos em grupos de trabalho é mais uma prática deste processo de engenharia de requisitos do que propriamente um passo da atividade *Assegurar uma Visão Comum*. A prática da comunicação no contexto da engenharia de software é considerada de grande importância em diversos estudos. Habilidades e atividades são citadas ao longo das áreas de processo de Desenvolvimento e Gerência de Requisitos do CMMI (SOFTWARE ENGINEERING INSTITUTE, 2002) e o primeiro valor básico que norteia as abordagens de XP e Modelagem Ágil é a comunicação (BECK, 2000), (AMBLER, 2002).

Uma avaliação empírica feita recentemente em um centro de desenvolvimento de software global (DAMIAN et al., 2004) concluiu que a prática de sessões de análise de requisitos em grupo contribui para uma melhor distribuição do conhecimento e uma melhora na comunicação entre desenvolvedores em 74% e 76%, respectivamente.

Realizar análise de requisitos em grupos significa organizar grupos de trabalho para atividades de engenharia de requisitos conforme a necessidade do projeto, com o principal objetivo de melhorar a comunicação entre os membros da equipe e com o cliente. Os grupos de trabalho são dinâmicos e têm como escopo diferentes níveis de requisitos do sistema.

Um exemplo de grupo de trabalho poderia ser composto por um analista, um projetista e um desenvolvedor para tratar da especificação de um conjunto de requisitos correlatos que representassem um módulo do sistema. Outro grupo poderia ser composto por um analista de sistemas, o gerente do projeto e o arquiteto de software, para trabalhar na priorização dos requisitos de uma iteração.

Estes grupos interagem em reuniões de freqüência predefinida. É comum, nas fases de Iniciação e Elaboração, que estas reuniões sejam diárias, com duração máxima de cerca de 30 minutos. Caso a agenda de uma das reuniões seja muito reduzida ou caso a equipe esteja envolvida em outras atividades naquele dia, a reunião pode ser cancelada ou pode ser apenas uma rápida

conversa informal, tratando dos principais assuntos pendentes. Um costume comum neste tipo de reunião é manter o foco em itens que precisam e podem ser resolvidos pelo grupo, deixando questões individuais e que precisam envolver outras pessoas registradas para tratamento fora da reunião.

A prática da análise de requisitos em grupos teve resultados excelentes nos projetos piloto observados durante este estudo.

4.3.2.3 Revisar artefatos

O processo de engenharia de requisitos proposto adota os conceitos de revisão de artefatos de requisitos apresentados em (LEFFINGWELL; WIDRIG, 2000). De acordo com estes conceitos, existem dois tipos de revisão: informal e formal. Esta atividade trata das revisões informais, feitas com os seguintes objetivos:

- Garantir a qualidade dos requisitos, identificando problemas de compreensão do negócio e das necessidades do cliente pela equipe de desenvolvimento.
- Difundir o conhecimento sobre o sistema, fazendo com que todos os envolvidos compreendam o sistema da mesma forma.
- Diminuir a dependência do projeto com relação a membros específicos da equipe e diminuir a necessidade de detalhamento da documentação por meio da promoção da comunicação entre a equipe.

As reuniões informais são feitas de acordo com a necessidade do projeto. Assim como as sessões de análise de requisitos em grupo, são planejadas sob demanda. Podem envolver apenas membros da equipe mas, em geral, pelo menos uma das revisões informais envolve também o cliente. A idéia é discutir a especificação de requisitos antes que esta esteja finalizada, antecipando o *feedback* do cliente.

Os artefatos sugeridos para revisão são:

- Documento de Visão;
- Especificação de Requisitos de Software (ERS);
- Especificação Funcional de Requisitos.

As revisões formais são tratadas no próximo passo da atividade de *Assegurar uma Visão Comum*.

4.3.2.4 Obter aprovações

É o resultado das revisões formais, ou seja, reuniões envolvendo o cliente que têm o objetivo principal de registrar a aprovação formal dos artefatos relacionados a requisitos.

O procedimento ideal é realizar uma revisão formal apenas após pelo menos uma revisão informal. Assim, qualquer ponto obscuro ou gerador de conflito já terá sido resolvido previamente e a aprovação será apenas uma formalidade. Contudo, para artefatos simples ou repetitivos, a aprovação pode ser pedida mesmo sem a realização de uma revisão informal.

As aprovações representam uma visão comum entre a equipe de desenvolvimento e o cliente, o registro do comprometimento da equipe com os requisitos traçados e o registro do comprometimento do cliente com os requisitos traçados.

As revisões não precisam ser feitas por meio de reunião. Geralmente, revisões informais são feitas com reuniões presenciais (ou via teleconferência) e revisões formais são feitas pelo cliente que, ao terminar, envia um e-mail oficializando a aprovação.

Os documentos listados a seguir precisam, obrigatoriamente, de aprovações do cliente:

- Documento de Visão;
- Especificação de Requisitos de Software (ERS);
- Especificação Funcional de Requisitos.

Estas aprovações delimitam o escopo do sistema, e permitem que o gerente de projeto realize a gerência de escopo e demais negociações necessárias no projeto com relação aos requisitos que devem fazer parte do sistema.

Atividades que terminam com a conclusão de um artefato só devem ser consideradas terminadas após a revisão e aprovação do mesmo.

4.3.2.5 Manter Glossário

O Glossário do sistema deve ser mantido desde as primeiras atividades da engenharia de requisitos. Seu objetivo é definir uma linguagem comum a todo o sistema, utilizando os termos de negócio familiares ao cliente, e difundindo estes termos para a equipe de desenvolvimento.

Sempre que um membro da equipe identificar um termo pertencente ao domínio do problema que o sistema se propõe a resolver, recomenda-se que esta pessoa registre o termo e uma breve descrição no Glossário.

4.4 Definir o Escopo do Sistema

Os objetivos do fluxo de trabalho de definição do escopo do sistema podem ser sumarizados da seguinte forma:

- Compreender o problema que deve ser resolvido pelo sistema, considerando o domínio do negócio do cliente.
- Identificar e documentar as necessidades do cliente e as características que o sistema deverá apresentar para atendê-las (requisitos do cliente).
- Identificar os critérios de aceitação destas características.
- Identificar as restrições impostas ao sistema por razões políticas, econômicas, técnicas ou de qualquer outra natureza.
- Identificar e documentar os requisitos de *software* do sistema para implementar as características (requisitos do produto).

O fluxo de trabalho *Definir o Escopo do Sistema* faz parte da fase de Iniciação do projeto, cujo objetivo é justamente o de caracterizar o escopo do sistema, identificando seus requisitos.

Para atingir os objetivos descritos acima, são executadas as seguintes atividades:

- Compreender Requisitos do Cliente
- Compreender Requisitos do Produto
- Gerenciar Requisitos
- Assegurar uma Visão Comum

Cada uma destas atividades é comentada nas próximas seções.

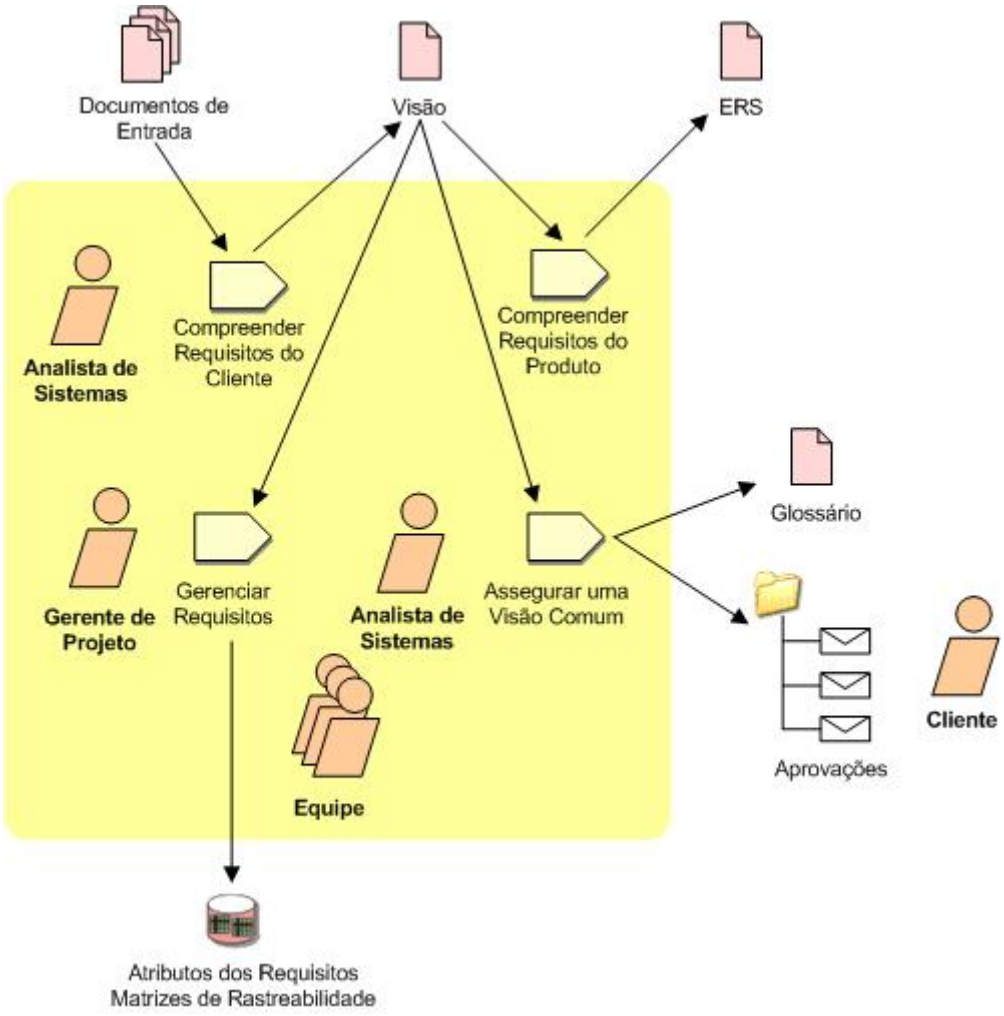


Figura 4.7: Definir o Escopo do Sistema

4.4.1 Compreender os Requisitos do Cliente

4.4.1.1 Objetivos, Papéis e Artefatos

A definição do sistema será baseada no problema a ser resolvido e nos *requisitos do cliente*, ou seja, nas necessidades, expectativas e restrições dos *stakeholders* (SOFTWARE ENGINEERING INSTITUTE, 2002).

Os objetivos da atividade de *Compreender os Requisitos do Cliente* são:

- Compreender o problema que deve ser resolvido pelo sistema.
- Levantar as necessidades dos envolvidos e as restrições impostas por estes.
- Identificar e documentar as características que o sistema deve apresentar para satisfazer estas necessidades.
- Analisar e harmonizar necessidades, características e restrições.

Para compreender o problema é preciso, primeiramente, identificar as pessoas capazes de fornecer informações sobre o negócio: os *stakeholders*. Em particular, estas pessoas devem poder informar o cenário percebido que oferece uma oportunidade de desenvolvimento de software. A seguir, deve-se interagir com os *stakeholders* para o levantamento, análise, discussão, revisão

e documentação das dificuldades identificadas neste cenário, as causas destas dificuldades e as necessidades e restrições dos *stakeholders* com relação ao sistema a ser desenvolvido. Partindo das necessidades identificadas, devem-se definir as características que o sistema deve apresentar para satisfazê-las. Todos estes itens representam os requisitos do cliente.

Os requisitos identificados podem ser incompletos, imprecisos, não consensuais e até mesmo conflitantes. Assim, é preciso analisá-los de forma a buscar acordos entre os *stakeholders* até que se alcance uma visão de comum acordo sobre os requisitos definitivos do sistema.

O responsável por esta atividade é o analista de sistemas. Para compreender os requisitos do cliente, o analista de sistema deve seguir os passos abaixo:

1. Identificar os fornecedores de requisitos;
2. Entender o problema;
3. Definir os limites do sistema;
4. Identificar as necessidades e restrições dos *stakeholders*.
5. Definir as características do sistema.

Todas estas informações são registradas no Documento de Visão.

4.4.1.2 Identificar os fornecedores de requisitos

Este é um passo crucial no entendimento do problema a ser resolvido a na definição do sistema que deve resolvê-lo.

Um dos conceitos básicos utilizados neste estudo é o termo *stakeholder*, ou seja, toda pessoa interessada no sistema e que será afetada por seu desenvolvimento. Os fornecedores de requisitos são *stakeholders* oficialmente designados pelo cliente como representantes de todas as equipes interessadas no projeto de desenvolvimento do sistema.

É importante trabalhar junto ao cliente na definição dos fornecedores de requisitos. Estes devem, em conjunto, possuir uma visão global do contexto onde o sistema se insere, desde o ponto de vista de negócios até o cotidiano dos usuários finais passando pelas equipes de tecnologia, suporte, treinamento e todos os outros setores da organização que estarão envolvidos na utilização do sistema ou que terão algum papel relacionado ao suporte do sistema.

Os fornecedores de requisitos devem, ainda, possuir autoridade suficiente para tomar decisões em nome dos setores que representam. Além disso, é importante definir uma hierarquia entre os fornecedores de requisitos, com um gerente de projeto e um gerente sênior (possivelmente a mesma pessoa), para os quais possam ser escaladas decisões sobre as quais não for possível chegar a um consenso.

Os fornecedores de requisitos devem ser registrados no Documento de Visão, conforme a tabela abaixo.

3. Fornecedores de Requisitos

[Entre os fornecedores de requisitos, devem ser identificados, obrigatoriamente, um Gerente de Projeto e um Gerente sênior, para os quais possam ser escaladas decisões sobre as quais não foi possível chegar a um consenso entre os demais fornecedores de requisitos.]

Nome	Responsabilidades	Critério de Designação	Dados para contato
[Informe o nome do fornecedor de requisitos]	[Faça uma breve descrição das atividades do fornecedor de requisitos e indique seu cargo.]	[Como o fornecedor de requisitos foi designado: - indicado pelo cliente - nomeado por fulano - etc.]	[Informe os dados para contato: e-mail, telefones, etc...].

Fornecedores de Requisitos

Nome	Responsabilidades	Critério de Designação	Dados para contato
João da Silva	– responsável pela área de vendas da empresa; – supervisiona as gerências das lojas e a seleção dos itens que são oferecidos.	Indicado pelo Presidente da Empresa	- Gerente de vendas - joao_silva@saber.com.br - Fone: 33339999, ramal 201
José dos Santos	– responsável pela implantação da solução	Nomeado pelo Sr. João da Silva	- Analista de Comércio Eletrônico - jose_santos@saber.com.br - Fone: 33339999, ramal 215
Antônio Oliveira	– responsável pela definição do processo de pagamento das compras online.	Indicado pelo Gerente Financeiro, Sr. Pedro Matos	- Analista de Cobrança - antonio_oliveira@saber.com.br - Fone: 33339999, ramal 211

Figura 4.8: Documento de Visão – Fornecedores de Requisitos

4.4.1.3 Entender o problema

Entender o problema significa obter o conhecimento de qual o real problema a ser resolvido (ou qual a oportunidade a ser implementada) por um sistema, e identificar qual seria a melhor solução para resolvê-lo.

> *Um problema pode ser definido como a diferença entre as coisas que são* ***percebidas*** *e as coisas que são* ***desejadas****.* (RATIONAL UNIVERSITY, 2002-b)

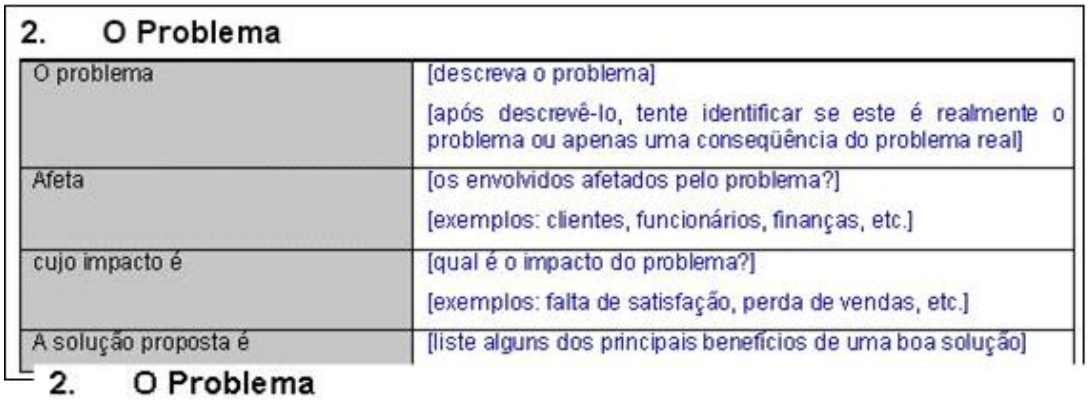
Partindo da definição acima, pode-se perceber que implementar todos os desejos manifestados pelo cliente não é, necessariamente, a forma pela qual se resolve o problema. Primeiramente, deve-se ter certeza de que a percepção do cliente sobre a situação atual é correta. Se a percepção do cliente estiver equivocada, pode ser que a situação atual esteja mais próxima do ideal do que ele imagina, tornando o problema menor ou mesmo inexistente.

Outra prática utilizada na resolução do problema é investigá-lo mais a fundo, com a participação do cliente, chegando mais perto de suas verdadeiras causas. Esta percepção pode mudar o que é desejado pelo cliente.

As verdadeiras causas, ou *causas-raiz* do problema, são as responsáveis por diversos sintomas que são percebidos pelos fornecedores de requisitos como problemas. Por exemplo, um problema pode ser “Os nossos clientes não estão satisfeitos com nossos serviços” ao passo que suas causas podem ser

“O nosso processo de pagamento é pouco ágil” ou “Há muitas falhas no nosso processo de vendas”.

Ao finalizar este passo de entendimento do problema, o analista de sistema deve ter condições de criar uma frase que defina o problema atacado e identifique os envolvidos e o impacto sofrido por eles. Deve, ainda, identificar uma possível solução para o problema. A figura abaixo ilustra a definição do problema e sugestão de solução que integram o Documento de Visão do processo de engenharia de requisitos proposto.



2. O Problema

O problema	[descreva o problema] [após descrevê-lo, tente identificar se este é realmente o problema ou apenas uma conseqüência do problema real]
Afeta	[os envolvidos afetados pelo problema?] [exemplos: clientes, funcionários, finanças, etc.]
cujo impacto é	[qual é o impacto do problema?] [exemplos: falta de satisfação, perda de vendas, etc.]
A solução proposta é	[liste alguns dos principais benefícios de uma boa solução]

2. O Problema

O problema	A Livraria Saber possui uma fatia importante do mercado tradicional, mas perde clientes que preferem adquirir livros pela internet.
Afeta	- Clientes, que precisam visitar a loja para comprar.
cujo impacto é	- Infidelidade dos clientes, que compram de outras livrarias que já possuem loja virtual - Perda de vendas, pois o percentual de vendas online está cada vez maior
A solução proposta é	Incluir no site da empresa um módulo de comércio eletrônico que permitirá aos clientes a aquisição de qualquer produto do estoque da loja sem sair de casa.

Figura 4.9: Documento de Visão – O Problema

4.4.1.4 Definir os limites do sistema

O sistema a ser desenvolvido se insere em um contexto, dentro da organização. Este contexto inclui, entre outros fatores, os funcionários que utilizarão o sistema, o pessoal de manutenção, repositórios de dados relacionados ao sistema (incluindo repositórios que o alimentam ou que utilizam dados gerados por ele), sistemas legados, mecanismos de comunicação dos quais o sistema se utilizará.

É importante definir claramente os limites do sistema e documentar que fatores do contexto organizacional fazem parte do sistema e quais estão fora de seu escopo.

Em (LEFFINGWELL; WIDRIG, 2000), algumas questões são sugeridas para auxiliar na definição do escopo do sistema:

- Quem usa o sistema?
- Quem fará manutenção no sistema?
- Quem recebe as saídas do sistema? (exemplo: Relatórios)
- Como o sistema comunica-se com outros sistemas?

Para documentar os limites do sistema, é criado um diagrama de perspectiva do produto que evidencia o contexto de inserção do sistema, seus usuários, sistemas legados e demais pontos de integração. O processo de engenharia de requisitos proposto não define a notação a ser utilizada no diagrama de perspectiva do produto, mas sugere a utilização de um diagrama UML evidenciando todos os atores do sistema.

A seção de definição dos limites do sistema no modelo do documento de Visão do processo proposto é ilustrada na figura a seguir.

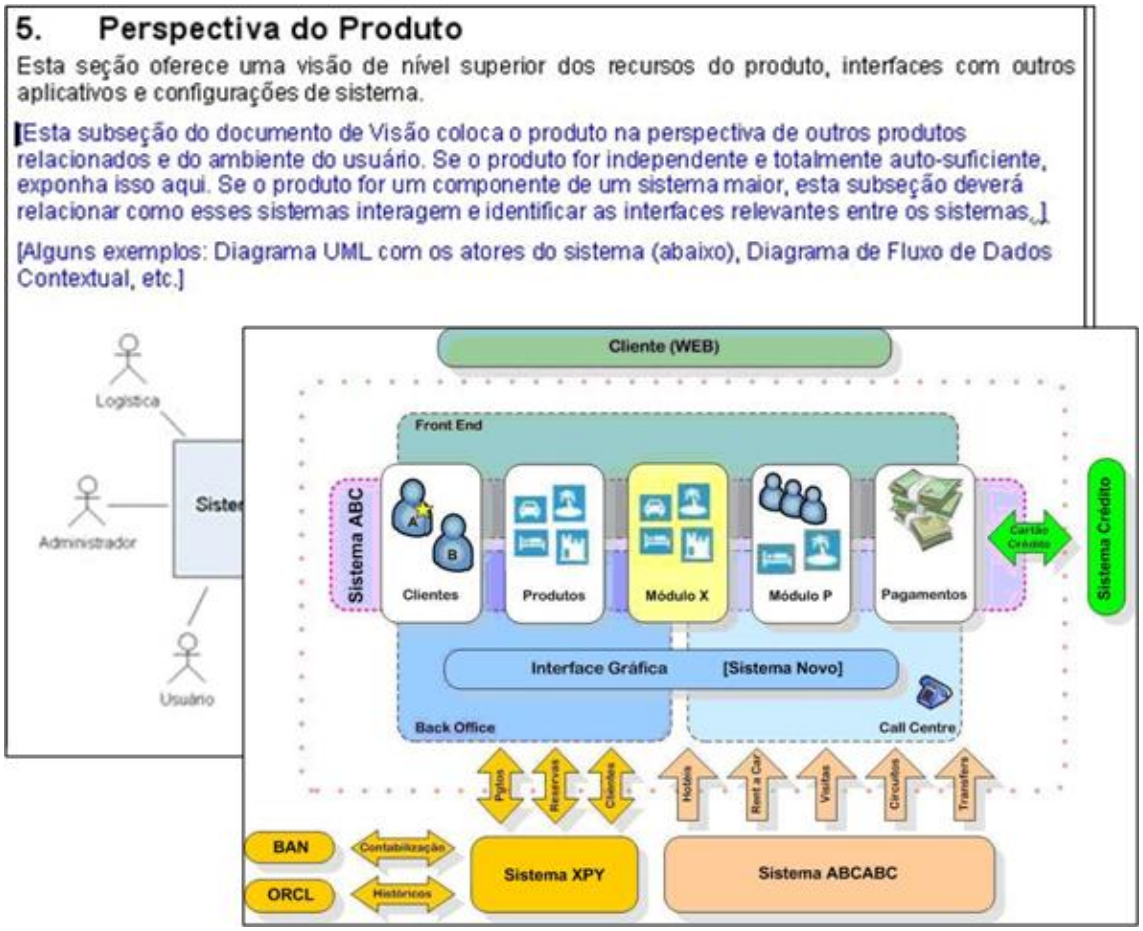


Figura 4.10: Documento de Visão – Perspectiva do Produto

4.4.1.5 Identificar as necessidades e restrições dos stakeholders.

Ao executar esta atividade, o analista de sistema identifica as necessidades do stakeholders e as restrições impostas por estes à solução do sistema.

Nesta etapa, o analista de sistema deve utilizar técnicas de elicitação de requisitos para obter a lista de necessidades, informadas pelos fornecedores de requisitos, que devem ser satisfeitas pelo sistema. Além disso, deve registrar as restrições informadas por eles.

As listas de necessidades e restrições levam algum tempo para se estabilizar devido, principalmente, aos seguintes fatores:

1. Conflitos entre requisitos: conforme mencionado anteriormente neste trabalho, o cliente pode manifestar requisitos contraditórios ao definir suas necessidades e restrições. Estes requisitos devem ser analisados, negociados com os fornecedores de requisitos e resolvidos, resultando em uma lista de necessidades e restrições correta e coerente.

2. Lacunas nas descrições: algumas necessidades ou restrições são informadas de forma vaga, imprecisa ou incompleta. Estas informações devem ser devidamente revisadas e completadas.

4.4.1.6 Definir as características do sistema

As características são serviços ou qualidades do sistema que satisfazem as necessidades dos *stakeholders*. Embora forneçam uma idéia de funcionalidade, as características devem ter como foco o funcionamento conceitual do sistema (*o que*) e não sua implementação (*como*).

As características são derivadas das necessidades e restrições e complementadas com requisitos decorrentes de restrições técnicas, de escolhas de arquitetura da solução ou outras considerações semelhantes.

É uma prática comum realizar a identificação de todos os requisitos do cliente (necessidades, restrições e características) de forma conjunta, já que estes requisitos tendem a ser identificados em momentos únicos.

As características são descritas no Documento de Visão, que serve de acordo com o cliente sobre o que é esperado do sistema. Assim, o nível de detalhe de cada característica deve ser geral o suficiente para que qualquer *stakeholder* possa entendê-la.

Estabelecer critérios objetivos de aceitação dos requisitos é parte da compreensão destes requisitos. Os critérios de aceitação são usados para confirmar que o sistema satisfaz as especificações solicitadas.

Caso um requisito não atinja algum de seus critérios de aceitação, o cliente pode rejeitá-lo com argumentos claros e irrefutáveis. Por outro lado, caso um requisito satisfaça todos os seus critérios de aceitação, a equipe de desenvolvimento tem a garantia de que o cliente não pode rejeitar o sistema alegando que deseja mais funcionalidades implementadas.

Os critérios de aceitação são definidos na etapa de Iniciação, durante a qual o escopo do sistema está sendo definido. São registrados no Documento de Visão, e podem ser associados a características ou ao sistema como um todo. A seguir é apresentada uma ilustração das seções de características e critérios de aceitação do modelo de Documento de Visão do processo proposto.

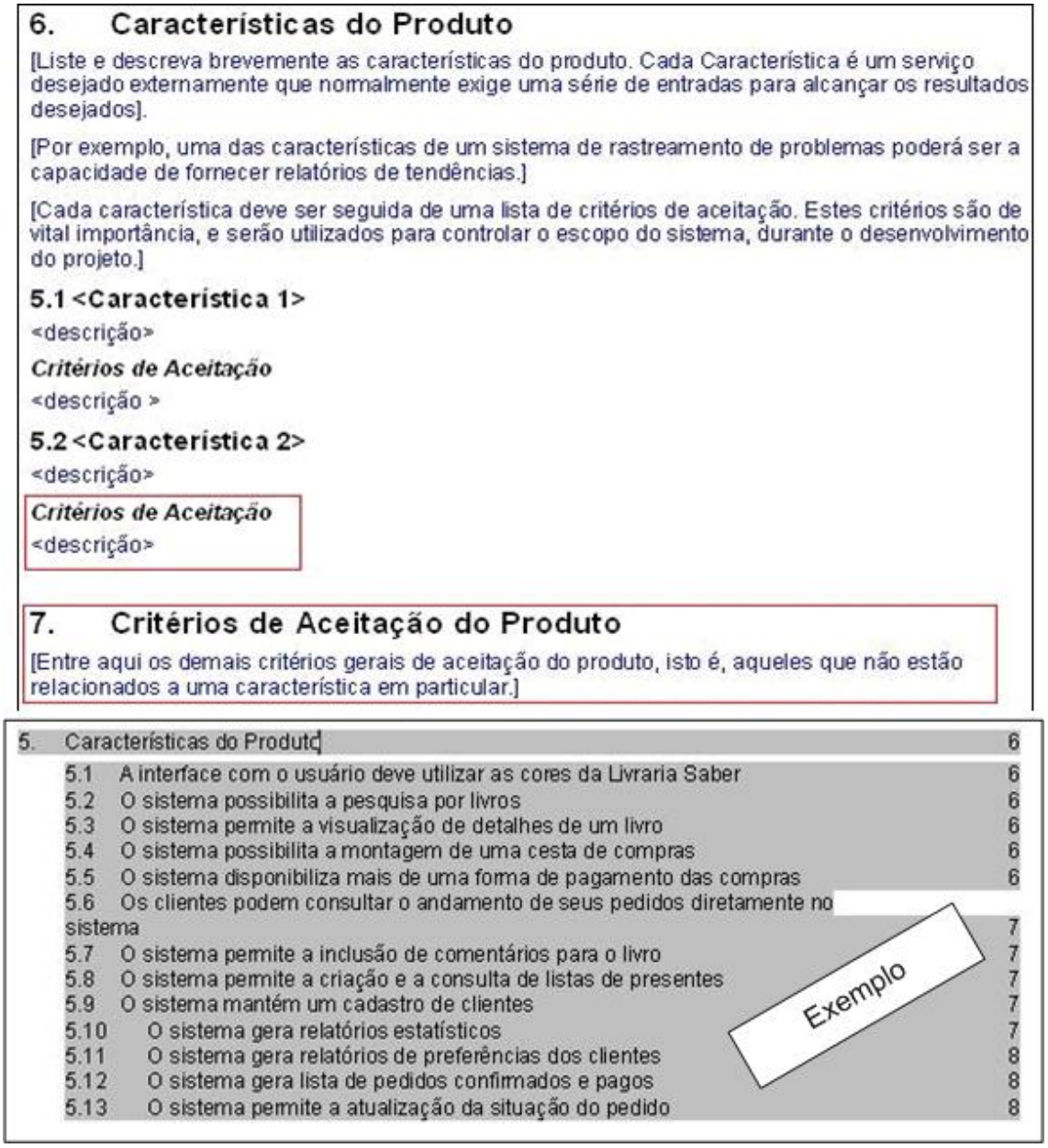


Figura 4.11: Documento de Visão – Características do Produto

4.4.2 Compreender os Requisitos do Produto

4.4.2.1 Objetivos, Papéis e Artefatos

Requisitos do cliente são refinados e elaborados para desenvolver requisitos de produto, ou seja, os requisitos de software do sistema, visíveis externamente.

O responsável pela compreensão de requisitos do produto é o analista de sistema, que deve seguir os passos abaixo:

1. Identificar Requisitos Funcionais e Não Funcionais;
2. Documentar Requisitos de Software.

Diversos artefatos de projeto são gerados durante a execução do primeiro passo. Porém, apenas a elaboração de um documento é obrigatória no processo de engenharia de requisitos proposto: o documento de *Especificação*

de Requisitos de Software (ERS), criado no segundo passo desta atividade de compreensão dos requisitos do produto.

4.4.2.2 Identificar Requisitos Funcionais e Não Funcionais

O conceito de requisito de software do sistema foi apresentado na sessão *Requisitos: Conceitos e Definições* deste trabalho, que também descreve os tipos de requisitos como FURPS+. Em resumo, esta sessão define que:

- Os requisitos de software podem ser funcionais (F) ou não funcionais (URPS+).
- Os requisitos funcionais descrevem a interação do sistema com seus atores (entidades externas ao sistema).
- Os requisitos não funcionais descrevem qualidades a serem observadas no sistema. Podem ser de usabilidade (U), confiabilidade (R, do inglês *realiability*), performance (P), suporte (S), além de outras categorias como restrições técnicas, requisitos de interface ou de *hardware*.

Estes requisitos são elaborados a partir das características do sistema, descritas no documento de Visão do Produto, e complementados através de técnicas de análise de requisitos.

Assim como adotado em RUP (IBM RATIONAL CORPORATION, 2003), o processo de engenharia de *software* proposto neste trabalho recomenda a utilização da modelagem de casos de uso como técnica de análise de requisitos para identificar requisitos funcionais e não funcionais do sistema. Embora esta técnica faça parte do processo proposto, estando inserida inclusive nos treinamentos adotados como parte deste processo, não será descrita neste trabalho, já que foi adotada na íntegra da forma como é descrita por seus criadores (JACOBSON et al., 1992), (BOOCH; RUMBAUGH; JACOBSON, 1999) e demais autores (ROSEMBERG, 1999), (ARMOUR; MILLER, 2001).

4.4.2.3 Documentar Requisitos de Software

Conforme a descrição dos artefatos apresentada no início deste capítulo, a Especificação de Requisitos de Software é composta pelo detalhamento de todos os requisitos de software do sistema, funcionais e não funcionais, e é documentada em forma de um pacote de artefatos. Já o documento de Especificação de Requisitos de Software (ERS) possui uma descrição geral das especificações funcionais e uma sessão de especificações suplementares do sistema.

Esta composição do documento ERS tem origem no modelo de ERS de RUP mas exclui as descrições detalhadas do documento. A figura abaixo ilustra o índice do modelo de documento ERS do processo de engenharia de requisitos proposto.

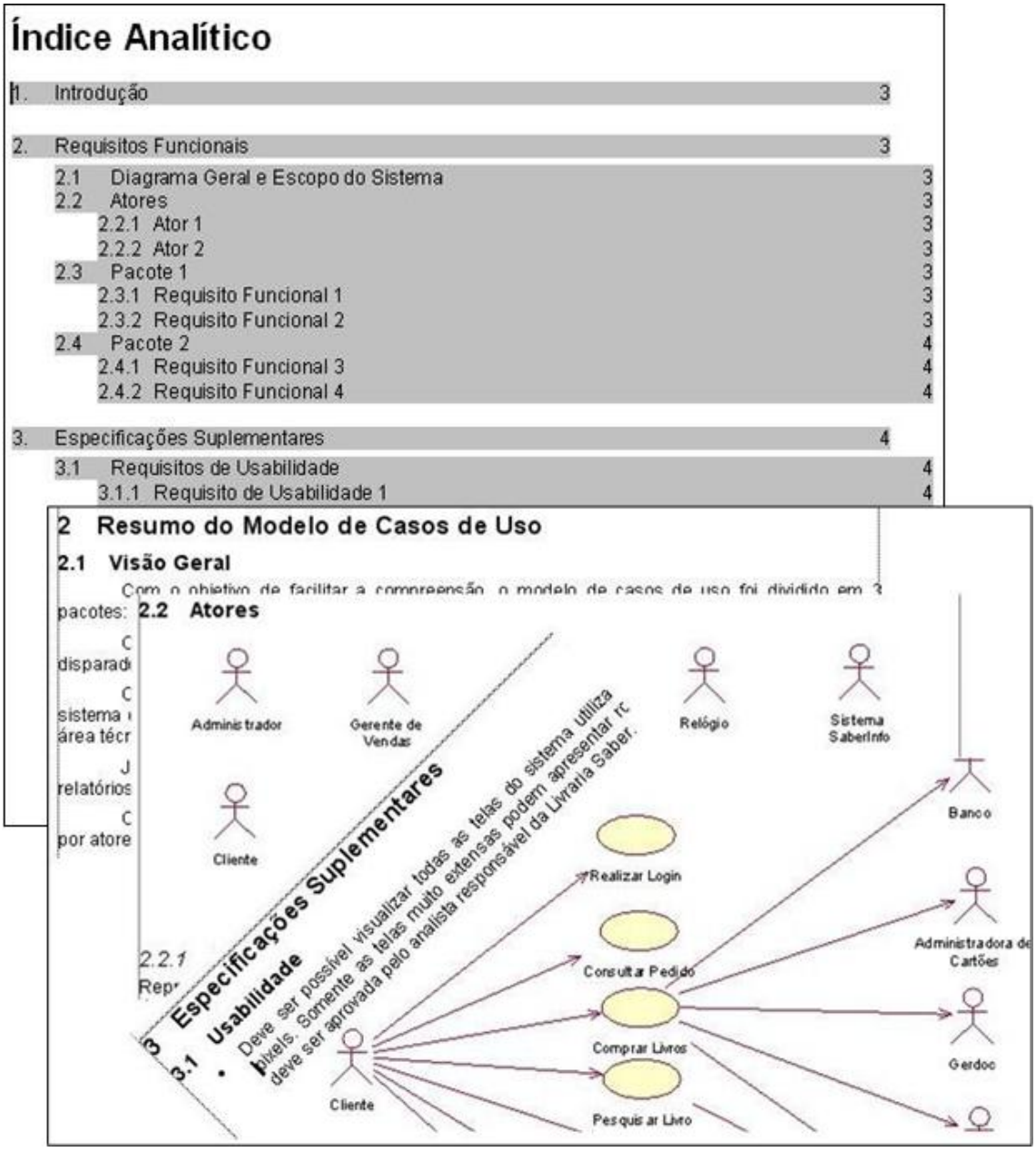


Figura 4.12: Índice do Modelo de Documento ERS

Quando a equipe de desenvolvimento opta pela utilização de modelagem de casos de uso, a documentação dos requisitos de software ocorre, freqüentemente, da seguinte forma:

- A identificação de requisitos funcionais e não funcionais (passo anterior) ocorre de forma conjunto, através da construção do modelo de casos de uso do sistema.
- O modelo de casos de uso inicial do sistema é criado através de um ou mais *workshops* de casos de uso. Após isso, novas sessões de modelagem e algumas especificações de caso de uso são feitas com o objetivo de estabilizar o modelo de casos de uso do sistema.
- O modelo de casos de uso do sistema é organizado em pacotes para melhor compreensão e manutenção do mesmo.

- Após a estabilização do modelo, a seção de *Requisitos Funcionais* do ERS é documentada, com breves descrições dos atores e casos de uso do sistema.
- Requisitos genéricos ou que não ficam propriamente representados através do modelo de casos de uso são descritos na seção de *Especificações Suplementares* do ERS.

Note-se que o modelo de casos de uso pode conter requisitos não funcionais, como por exemplo o tempo máximo de resposta durante a execução de uma transação. Nestes casos, os requisitos não funcionais não aparecem de forma explícita no documento ERS, sendo documentados apenas nas especificações funcionais de casos de uso, no fluxo de atividades *Refinar Requisitos de Software*.

4.4.3 Gerenciar Requisitos

Esta atividade, recorrente em vários fluxos de trabalho, já foi descrita anteriormente (ver seção *Atividades Recorrentes*). Contudo, no fluxo de trabalho *Definir o Escopo do Sistema*, cada passo da gerência de requisitos é executado de forma específica:

1. Manter atributos dos requisitos: são registrados todos os atributos associados a cada necessidade, característica e requisitos de software do sistema. Os atributos relacionados a priorização (relevância para o negócio, relevância para a arquitetura e prioridade) ainda não são definidos.
2. Manter rastreabilidade: são preenchidas as seguintes matrizes de rastreabilidade:
   - Necessidades x Características
   - Características x Requisitos Funcionais
   - Características x Requisitos Não Funcionais
3. Priorizar requisitos: o atributo de prioridade dos requisitos funcionais é preenchido. Os demais atributos relacionados à prioridade dos requisitos do sistema são opcionais.

4.4.4 Assegurar uma Visão Comum

Esta atividade, recorrente em vários fluxos de trabalho, já foi descrita anteriormente (ver seção *Atividades Recorrentes*). Neste fluxo de trabalho, cada passo executado mantém o foco na definição de escopo do sistema:

1. Realizar análise de requisitos em grupos: sessões de análise em grupo típicas deste fluxo de trabalho são: discussão do problema, identificação de necessidades e características; modelagem de casos de uso.
2. Revisar artefatos e Obter aprovações: revisões informais e formais são feitas sobre o documento de Visão e o documento ERS.
3. Manter Glossário: este artefato é criado e as primeiras definições soa registradas. Diversos termos do Glossário são identificados durante as sessões de análise de requisitos em grupo.

4.4.5 Resumo do Fluxo de Trabalho

Tabela 4.3: Definir o Escopo do Sistema – Resumo

Fluxo de Trabalho: Definir o Escopo do Sistema
Fase de Execução: Iniciação
Responsáveis: Equipe, Analista de Sistemas, Gerente de Projeto, Cliente
Artefatos: • Documentos de Entrada • Documento de Visão • Glossário • Especificação de Requisitos de Software (ERS) • Atributos de Requisitos • Matrizes de Rastreabilidade
Atividades:

• Compreender os requisitos do cliente
  - Identificar fornecedores de requisitos
  - Entender o problema
  - Definir os limites do sistema
  - Identificar as necessidades e restrições dos *stakeholders*
  - Definir as características do sistema
• Compreender os requisitos do produto
  - Identificar requisitos funcionais e não funcionais
  - Documentar requisitos de software
• Gerenciar requisitos
  - Manter atributos dos requisitos
  - Manter rastreabilidade
  - Priorizar requisitos
• Assegurar uma visão comum
  - Realizar análise de requisitos em grupo
  - Revisar artefatos
  - Obter aprovações
  - Manter glossário

Critérios de conclusão:

• Glossário iniciado
• Documento de visão aprovado
• Documento de Especificação de Requisitos de Software (ERS) aprovado
• Atributos dos requisitos definidos
• Matrizes de rastreabilidade preenchidas:
  - Necessidades x Características
  - Características x Requisitos Funcionais
  - Características x Requisitos Não Funcionais

4.5 Refinar Requisitos de Software

Este fluxo de trabalho se dedica ao refinamento dos requisitos identificados durante a definição do escopo do sistema. Este refinamento envolve a

documentação de informações detalhadas sobre os requisitos de software, a identificação de possibilidades de reutilização de componentes de análise de requisitos, a caracterização da interface do sistema com seus usuários e os primeiros esforços no sentido de criar o modelo de dados do sistema.

O refinamento de requisitos de software consolida o conhecimento adquirido sobre estes requisitos, evidencia pontos de falha e revela requisitos que possivelmente ainda não foram descobertos. Assim, este fluxo de trabalho se repete ao longo do processo, nas fases de Iniciação (opcionalmente), Elaboração (intensamente) e Construção.

As atividades envolvidas no fluxo de trabalho *Refinar Requisitos de Software* são:

- Especificar Requisitos de Software;
- Modelar a Interface;
- Analisar o Domínio;
- Gerenciar Requisitos;
- Assegurar uma Visão Comum.

Estas atividades são descritas a seguir,

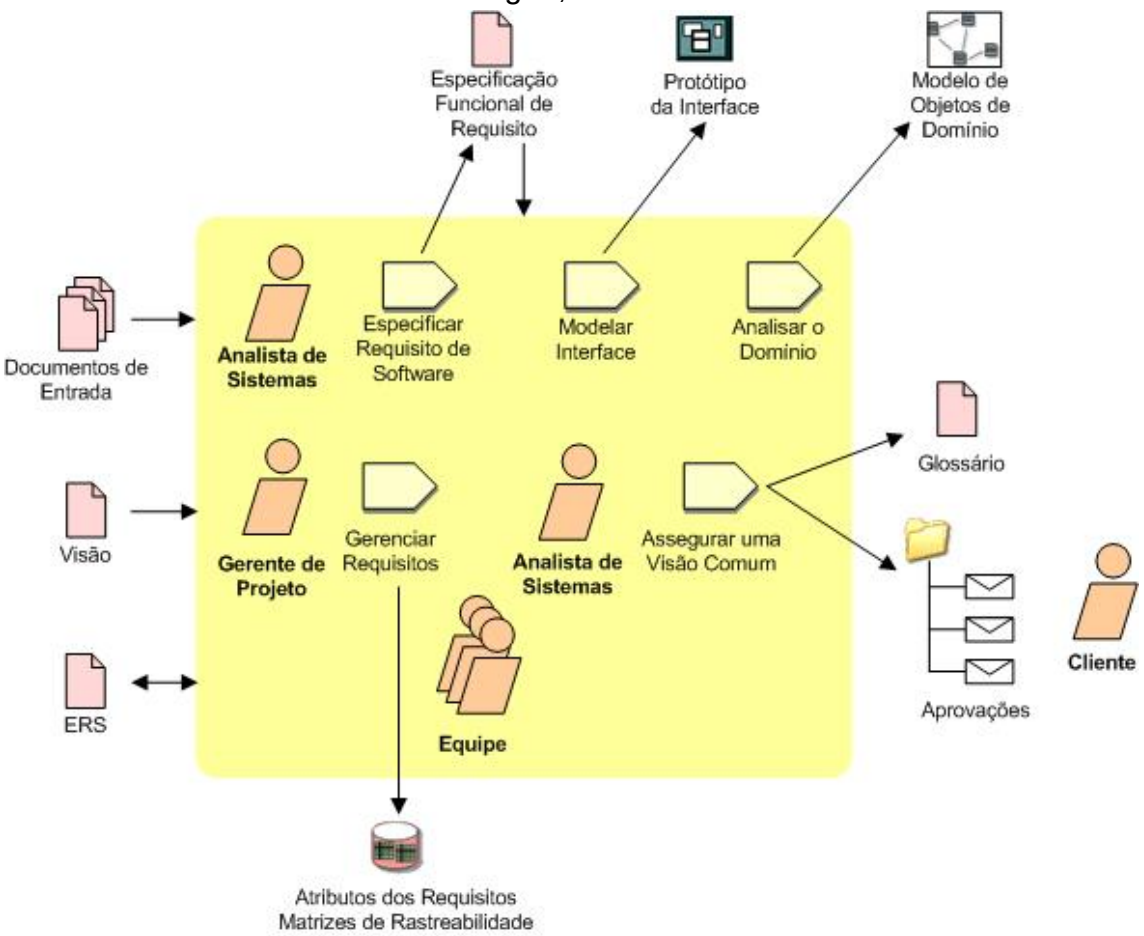


Figura 4.13: Refinar Requisitos de Software

4.5.1 Especificar Requisitos de Software

4.5.1.1 Objetivos, Papéis e Artefatos

O principal objetivo desta atividade é detalhar requisitos de software identificados durante a definição do escopo do sistema. Estes requisitos podem ser funcionais ou não funcionais.

O processo de engenharia de requisitos proposto sugere dois locais para o detalhamento de requisitos: a *Especificação Funcional de Requisitos* e a seção de *Especificações Suplementares* do documento ERS. Porém, a utilização de outros artefatos é aceitável, desde a documentação detalhada dos requisitos funcionais e não funcionais do sistema seja produzida.

Esta atividade se divide nos seguintes passos:

1. Detalhar requisitos de software;
2. Refinar modelos de análise;
3. Atualizar artefatos sobre requisitos.

O responsável por esta atividade é o analista de sistemas. Para executar os passos acima, o analista utiliza as informações do sistema registradas nos Documentos de Entrada, no documento de Visão e no documento ERS e gera Especificações Funcionais de Requisitos e possíveis atualizações das Especificações Suplementares.

Cada um dos passos listados acima é descrito a seguir.

4.5.1.2 Detalhar Requisitos de Software

Os requisitos de software do sistema, identificados em etapas anteriores de análise de requisitos, devem ser especificados em detalhe e documentados. A técnica de especificação sugerida é a da modelagem de casos de uso, que descreve, para cada caso de uso do modelo, os seguintes aspectos:

- *Descrição breve*: um pequeno parágrafo (geralmente um ou duas linhas) descrevendo o caso de uso. Esta descrição também aparece no documento ERS, e é feita ainda na fase de Iniciação do projeto, quando a equipe está trabalhando na definição de escopo.
- *Fluxo de eventos básico*: descreve a interação entre os atores e o sistema, na forma de um diálogo: "*O ator faz... O sistema faz ...*". Os eventos são descritos da forma mais independente possível da solução técnica, para evitar a tomada de alguma decisão que deveria ser feita durante as atividades de projeto (*design*) do sistema. O fluxo básico descreve os eventos que ocorrem mais freqüentemente, sem mencionar alternativas ou tratamento de erros.
- *Fluxos de eventos alternativos*: descreve os eventos alternativos que não aparecem no fluxo básico. Também assume a forma de um diálogo entre ator e sistema.
- *Fluxos de eventos de exceção*: descreve o tratamento de erro e os eventos de exceção que não aparecem no fluxo básico. Também assume a forma de um diálogo entre ator e sistema.
- *Regras de negócio*: são requisitos de software cuja descrição não é adequada para inserção nos fluxos de eventos: regras de negócio complexas, requisitos não funcionais, detalhes sobre algum objeto do domínio dos casos de uso.

- *Protótipo de interface com o usuário*: esta seção não aparece no modelo clássico de especificação de caso de uso. É opcional, e relaciona-se à atividade de *Modelar a Interface* do sistema. O objetivo desta seção é facilitar o entendimento do fluxo de eventos do caso de uso, fornecendo informações mais tangíveis do que o diálogo entre ator e sistema. Diversas técnicas podem ser utilizadas para preencher esta seção: *screenshots* de protótipos funcionais, representações de telas do sistema feitas com variadas ferramentas gráficas ou até mesmo desenhos feitos à mão e capturados (*storyboarding*). É importante que os envolvidos na elaboração e revisão deste documento, sobretudo o cliente, saibam que o protótipo de interface é apenas um facilitador, e não representa necessariamente todos os detalhes da tela final do sistema.

A especificação de um caso de uso pode ter mais de um requisito funcional e pode, inclusive, descrever requisitos não funcionais, caso estes se apliquem apenas ao caso de uso especificado. Por exemplo, a especificação do caso de uso *Comprar Produto* pode conter o seguinte requisito não funcional: "*O processamento da compra e o envio de uma resposta ao cliente deve ocorrer em no máximo 5 segundos*".

Ao detalhar casos de uso, o analista de sistemas identifica alguns requisitos (funcionais ou não) que não podem ser propriamente descritos na especificação funcional de requisitos. Isto acontece principalmente com requisitos não funcionais genéricos, que devem ser associados a todos os casos de uso do sistema. Nestes casos, estes requisitos são descritos na seção de *Especificações Suplementares* do documento ERS.

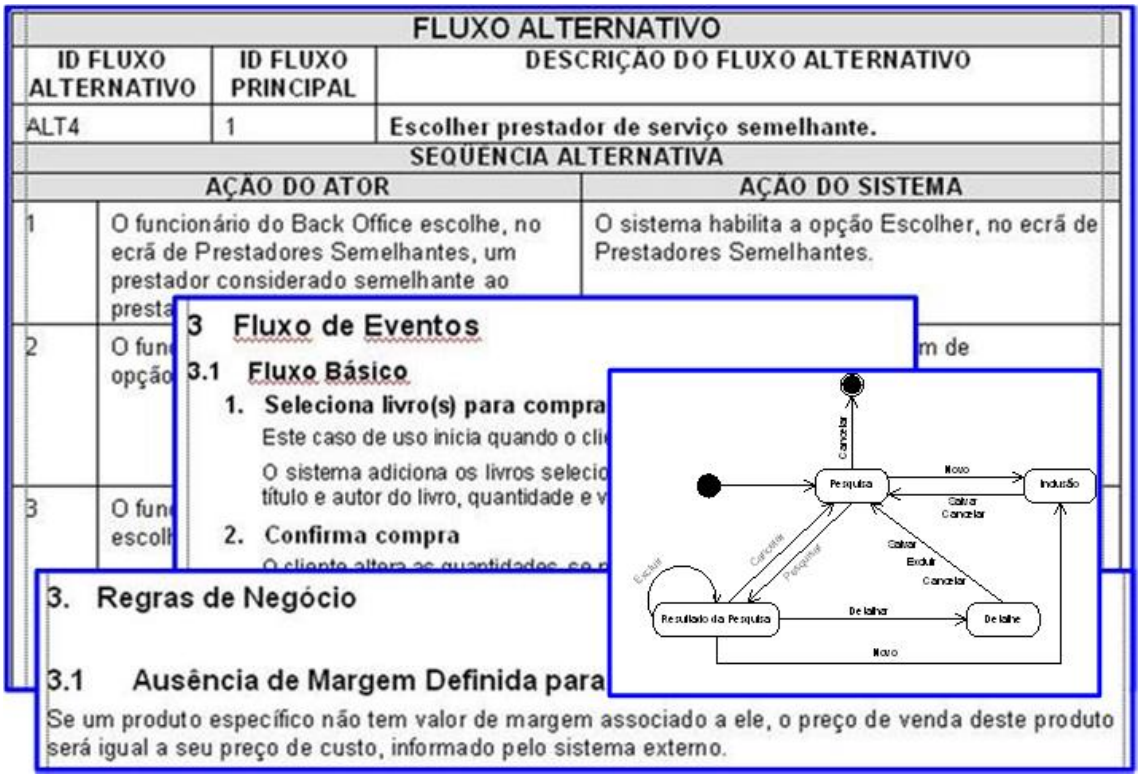


Figura 4.14: Refinar Requisitos de Software

4.5.1.3 Refinar Modelos de Análise

Com o detalhamento gradativo dos requisitos de software, o conhecimento sobre o modelo conceitual do sistema evolui, o que permite Identificar oportunidades de reuso e refinamento nos modelos de análise construídos. Novos requisitos são descobertos, outros são unificados ou corrigidos. Estas são as ações tomadas neste passo da especificação de requisitos.

Para projetos que utilizam a modelagem de casos de uso, isto significa estruturar o modelo criando relações entre casos de uso como *include*, *extend* ou *generalização*.

4.5.1.4 Atualizar artefatos sobre requisitos

Assim como os modelos de análise são refinados, alguns artefatos sobre requisitos também podem precisar de atualizações. Um exemplo freqüente é o documento de Especificação de Requisitos de Software, sobretudo a seção de *Especificações Suplementares*.

Note-se que todo artefato modificado após aprovação precisa ser revisado e oficialmente provado novamente.

4.5.2 Modelar Interface

4.5.2.1 Objetivos, Papéis e Artefatos

A interface com o usuário é um dos recursos mais produtivos na obtenção de *feedback* do cliente sobre o sistema em desenvolvimento. Por este motivo são inseridos protótipos de interface nas especificações funcionais de requisitos.

Além disso, sistemas de software precisam de interfaces planejadas e uniformes, para melhorar sua usabilidade, a facilidade de manutenção e até mesmo a produtividade, através do reuso de código padronizado.

Os parágrafos acima descrevem os objetivos da atividade *Modelar Interface*. O responsável por esta atividade é o analista de sistemas, que pode utilizar recursos como ferramentas de construção de protótipos, padrões de interface de mercado ou mesmo a ajuda de um *expert* em projeto gráfico.

O artefato gerado é um *Protótipo de Interface*, que pode ser: um documento textual sobre o comportamento, *layout* e outras características comuns da interface; um protótipo gráfico; ou qualquer outro formato que oriente a equipe de desenvolvimento sobre a padronização de interface.

Os protótipos de interface inseridos em especificações funcionais de requisitos são geralmente baseados neste protótipo de interface genérico.

4.5.3 Analisar o Domínio

4.5.3.1 Objetivos, Papéis e Artefatos

A análise de domínio tem o objetivo de desenvolver o modelo de objetos inicial do sistema (Modelo de Domínio), representando objetos tangíveis do domínio do problema. Esta atividade reforça o vocabulário comum promovido pelo Glossário do sistema.

Além do Glossário, diversos outros artefatos sobre requisitos servem como fonte para a análise de domínio: Documentos de Entrada, Visão, ERS,

Especificações Funcionais de Requisitos. O responsável é o analista de sistemas.

O Modelo de Domínio contribui com as atividades projeto (*design*) e modelagem de dados do sistema.

4.5.4 Gerenciar Requisitos

Esta atividade já foi descrita anteriormente (ver seção *Atividades Recorrentes*). Contudo, no fluxo de trabalho *Refinar Requisitos de Software*, a principal ação é o preenchimento da matriz de rastreabilidade de *Requisitos Funcionais x Especificações Funcionais*.

4.5.5 Assegurar uma Visão Comum

Esta também é uma atividade recorrente (ver *Atividades Recorrentes*). Neste fluxo de trabalho, sessões de análise em grupo típicas são revisões informais de especificações funcionais, onde dúvidas sobre o fluxo de eventos, pré e pós-condições são comentadas entre os analistas de sistemas. Cada Especificação Funcional de Requisito deve ser formalmente aprovada, assim como atualizações das Especificações Suplementares. O Glossário continua em evolução.

4.5.6 Resumo do Fluxo de Trabalho

Tabela 4.4: Refinar Requisitos de Software – Resumo

Fluxo de Trabalho: Refinar Requisitos de Software
Fase de Execução: Iniciação, Elaboração, Construção
Responsáveis: Equipe, Analista de Sistemas, Gerente de Projeto, Cliente
Artefatos: • Documentos de Entrada • Documento de Visão • Especificação de Requisitos de Software (ERS) • Glossário • Atributos de Requisitos • Matrizes de Rastreabilidade • Especificação Funcional de Requisito • Protótipo de Interface • Modelo de Objetos do Domínio
Atividades:

• Especificar Requisitos de Software
  - Detalhar requisitos de software
  - Refinar modelos de análise
  - Atualizar artefatos sobre requisitos
• Modelar a interface
• Analisar o domínio
• Gerenciar requisitos

- Manter atributos dos requisitos
- Manter rastreabilidade
- Priorizar requisitos

- Assegurar uma visão comum
  - Realizar análise de requisitos em grupo
  - Revisar artefatos
  - Obter aprovações
  - Manter glossário

Critérios de conclusão:

- Documento de visão aprovado
- Especificações Funcionais de Requisitos (de cada iteração) aprovadas
- Matriz de rastreabilidade Requisitos Funcionais x Especificações Funcionais preenchida.
- Glossário atualizado

4.6 Gerenciar Mudanças

> "*Práticas modernas de desenvolvimento de software reconhecem que especificações de requisitos estarão erradas em algum ponto constantemente, e identificam formas de mitigar este risco ao longo do projeto.*" (SOFTWARE ENGINEERING INSTITUTE, 2003)

A seção de *Engenharia de Requisitos: Definições e Problemas Clássicos*, apresentada neste trabalho, descreve os fatores geradores de mudanças em projetos de software: mudanças no negócio, descobertas sobre complexidade de requisitos, melhorias, evolução da percepção do funcionamento do sistema.

Conforme foi descrito nessa seção, as mudanças devem ser gerenciadas para que não se perca o controle sobre as variáveis de escopo, prazo e custo do projeto.

Do ponto de vista da engenharia de requisitos, as contribuições dadas à gerência de mudanças são:

- Documentar os requisitos, seus atributos e relações de dependências, priorização e histórico.
- Utilizar estes atributos para avaliar o impacto de mudanças sobre requisitos em termos de esforço.
- Validar, com o cliente, os resultados de alterações nos artefatos sobre requisitos decorrentes de mudanças no projeto.

O primeiro item da lista de contribuições acima, relacionado à documentação de atributos de requisitos, é coberto pela atividade *Gerenciar Requisitos*, presente em todos os fluxos de trabalho do processo de engenharia de requisitos proposto (ver seção de *Atividades Recorrentes*).

O último item dessa lista, a validação de requisitos, é satisfeito pela atividade *Assegurar uma Visão Comum*, responsável pela revisão e aprovação de artefatos sobre requisitos, executada em todos os fluxos de trabalho deste processo (ver seção de *Atividades Recorrentes*).

Assim, a única contribuição da engenharia de requisitos ao controle de mudanças no projeto que ainda não foi abordada é a utilização dos atributos de requisitos para avaliar o impacto das mudanças. Esta avaliação faz parte do processo de gerência de mudanças. Embora este processo não seja vinculado diretamente aos requisitos, já que controla mudanças em qualquer parte do projeto, será apresentado neste trabalho para evidenciar o tratamento de mudanças em requisitos.

Os principais objetivos da gerência de mudança, sob o enfoque da engenharia de requisitos, são:

- Registrar as mudanças;
- Analisar o impacto nos requisitos;
- Decidir sobre a adoção da mudança;
- Implementar a mudança.

A implementação da mudança exige o re-planejamento do projeto, atividade que faz parte dos fluxos de trabalho da gerência de projeto e, portanto, não será abordada neste trabalho. Da mesma forma, a definição da data de execução da mudança e o acompanhamento do esforço real necessários também fazem parte de outros fluxos de trabalho e não serão descritos aqui.

Os demais objetivos são alcançados pelas atividades do fluxo de trabalho Gerenciar Mudanças, detalhadas nas seções seguintes:

- Submeter solicitação de mudança;
- Complementar solicitação de mudança;
- Analisar solicitação de mudança.

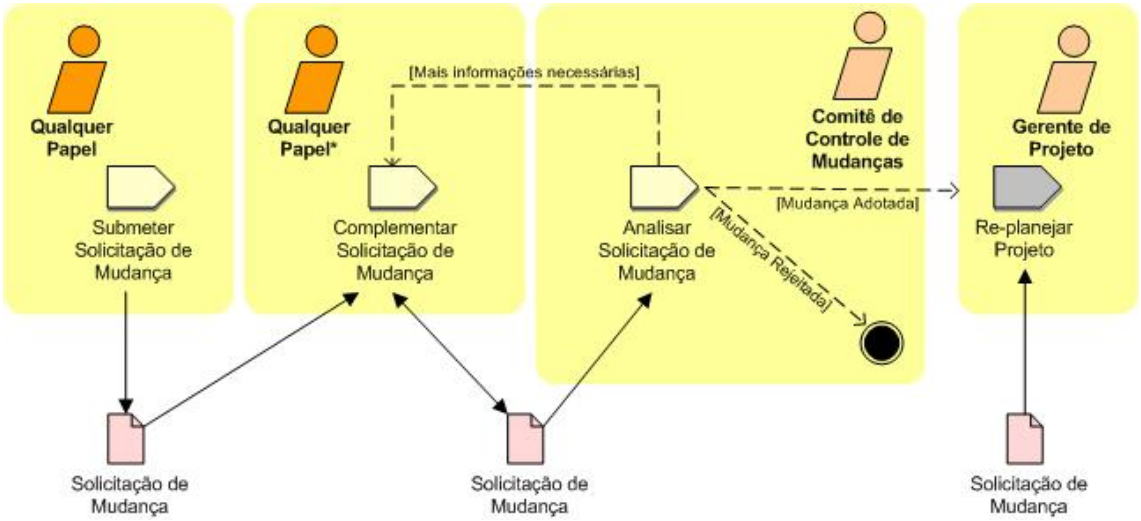


Figura 4.15: Gerenciar Mudanças

O artefato utilizado em todas estas atividades é a *Solicitação de Mudanças*. Este documento pode ser criado como um simples e-mail e deve evoluir ao longo do fluxo de trabalho até que esteja completo. A Solicitação de Mudança contém todas as informações pertinentes à mudança, tais como: identificação, status, descrição da situação atual e do problema que a mudança procura resolver, descrição da mudança, custo estimado de implementação da mudança, observações e informações resultantes da análise da mudança.

Note-se que, ao utilizar o termo *implementação da mudança*, não é feita uma referência ao código do sistema, mas sim a todos os artefatos onde a mudança deve ocorrer, inclusive artefatos relacionados a requisitos (Visão, ERS, Especificações Funcionais, etc.).

4.6.1 Submeter Solicitação de Mudança

4.6.1.1 Objetivos, Papéis e Artefatos

Diversas pessoas estão envolvidas no desenvolvimento de um sistema de software, tanto na organização cliente (participante externo) quando na equipe de desenvolvimento (participante interno). Quando um destes participantes percebe a necessidade de uma mudança no sistema, pode submeter uma solicitação de mudança.

A necessidade de mudança pode resultar de diversos eventos como a percepção de um erro de análise, de problemas encontrados nos sistema, de novas necessidades dos participantes externos ou de oportunidades de melhorias.

O artefato utilizado nesta atividade é a Solicitação de Mudança. Contudo, o formato inicial da solicitação pode ser até mesmo um e-mail descrevendo o problema encontrado e a sugestão de modificação. Os dados cadastrais da solicitação também devem ser preenchidos, conforme a ilustração abaixo.

Data:			
Identificador:	<identificador único da solicitação de mudança>		
Status:	☐ Submetida	☐ Em complementação ☐ Em aprovação	☐ Aprovada ☐ Rejeitada
Tipo:	☐ Defeito	☐ Melhoria	☐ Novo Requisito
DBserver:	**Gerente do Projeto**	<Nome do Gerente do Projeto>	
Solicitante:	<INTERNO x EXTERNO> <Nome, papel na estrutura do projeto>		
Comitê de Controle de Mudanças:	<INTERNO x EXTERNO> <Nome, papel na estrutura do projeto> <INTERNO x EXTERNO> <Nome, papel na estrutura do projeto> <INTERNO x EXTERNO> <Nome, papel na estrutura do projeto>		

Figura 4.16: Solicitação de Mudança – Dados Cadastrais

Conforme descrito acima, qualquer participante pode disparar uma Solicitação de Mudança. Esta deve ser endereçada ao Gerente de Projeto. Uma mesma Solicitação de Mudança pode ser composta por várias mudanças relacionadas, que devem ser analisadas em conjunto.

Primeiramente, as mudanças devem ser contextualizadas, descrevendo-se a que parte do sistema (módulo, programa, camada, etc.) estas se referem. Após isto, deve-se descrever a situação atual, com problemas, e as mudanças necessárias para resolvê-los (ver figura abaixo).

1. Contextualização das Mudanças

[Descrição do contexto do sistema onde as mudanças se inserem.]

2. Descrição das Mudanças

[Descrição do contexto do sistema onde as mudanças se inserem.]

[A descrição em forma de tabela é apenas uma sugestão. Podem ser incluídas capturas de telas, protótipos, *storyborting*, modelo, diagramas, ou qualquer outro recursos gráfico que facilite a descrição da situação atual e das mudanças propostas.]

Id	Situação Atual	Proposta
	[Descrição do problema enfrentado,; caracterização da falha, melhoria ou nova necessidade, com severidade e prioridade; condições sob as quais o problema foi detectado, etc.]	[Descrição da modificação proposta]
1		
2		
3		

Figura 4.17: Solicitação de Mudança – Descrição

4.6.2 Complementar Solicitação de Mudança

4.6.2.1 Objetivos, Papéis e Artefatos

Logo após submetida, uma Solicitação de Mudança contém apenas uma descrição inicial das mudanças desejadas (figura acima). O próximo passo é complementar a solicitação de forma que se possa decidir se as mudanças serão aceitas ou rejeitadas.

Esta complementação pode ser feita por diferentes participantes do projeto. Se as mudanças são relacionadas a requisitos, o responsável é o analista de sistemas, que deve executar dois passos:

1. Complementar a descrição da mudança
2. Analisar o impacto da mudança.

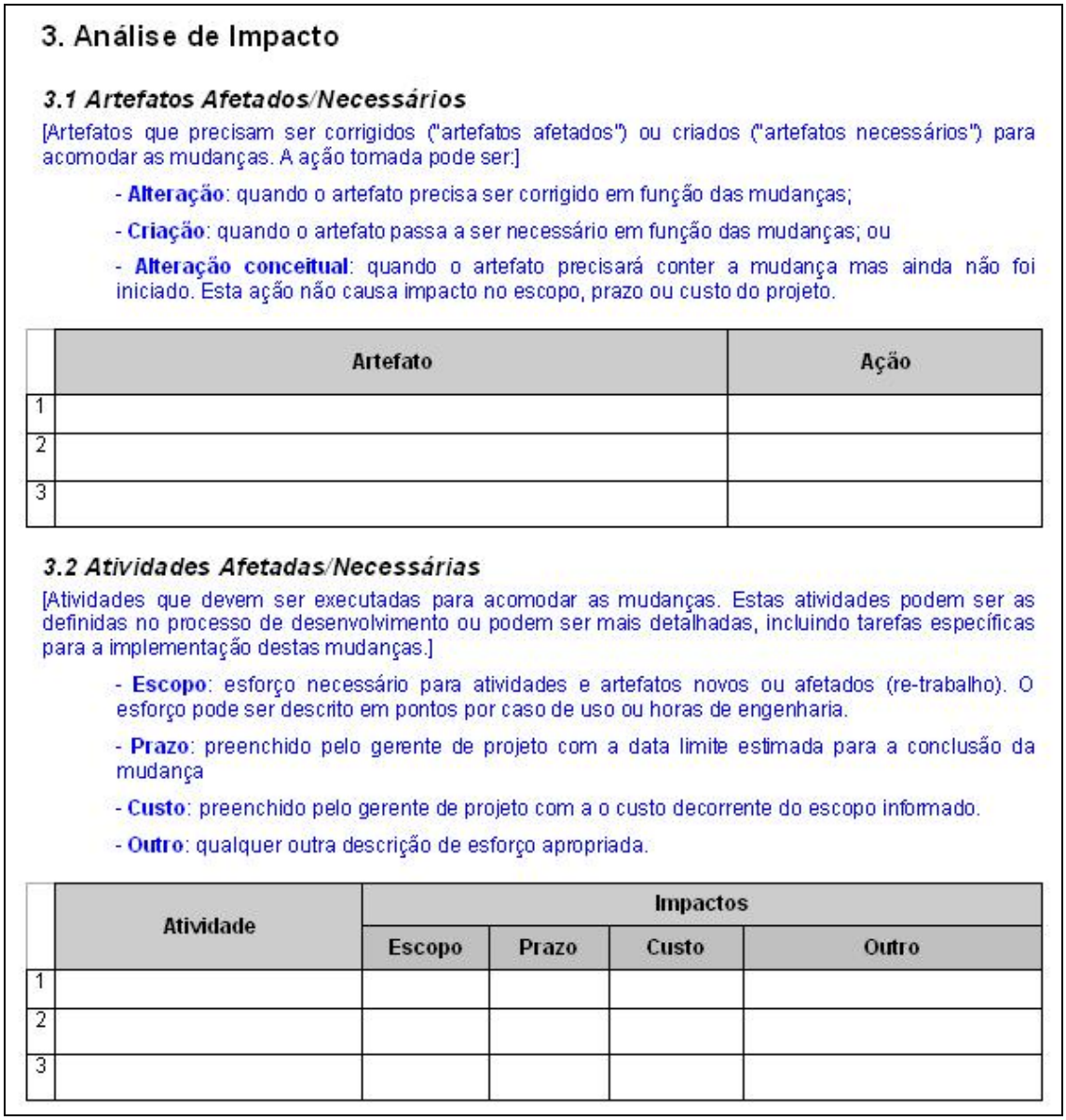
4.6.2.2 Complementar a Descrição da Mudança

Neste passo o analista de sistemas deve revisar a mudança, verificar se a descrição está clara e completa, certificar-se de que o problema não pode ser resolvido no sistema atual e de que não existe nenhuma outra solicitação de mudança que contemple as mudanças da solicitação analisada.

Quando necessário, o analista de sistemas deve completar a descrição, possivelmente com a ajuda do solicitante, e inserir os comentários apropriados.

4.6.2.3 Analisar o Impacto da Mudança

Compreender e estimar as conseqüências da adoção das mudanças para o projeto, identificando os artefatos afetados e as atividades necessárias para implantas as mudanças. Um recurso que auxilia na identificação de artefatos afetados é observar a documentação sobre rastreabilidade. Para cada atividade, deve ser feita uma estimativa de impacto no escopo, prazo e custo do projeto.



3. Análise de Impacto

3.1 Artefatos Afetados/Necessários

[Artefatos que precisam ser corrigidos ("artefatos afetados") ou criados ("artefatos necessários") para acomodar as mudanças. A ação tomada pode ser:]

- **Alteração**: quando o artefato precisa ser corrigido em função das mudanças;
- **Criação**: quando o artefato passa a ser necessário em função das mudanças; ou
- **Alteração conceitual**: quando o artefato precisará conter a mudança mas ainda não foi iniciado. Esta ação não causa impacto no escopo, prazo ou custo do projeto.

	Artefato	Ação
1		
2		
3		

3.2 Atividades Afetadas/Necessárias

[Atividades que devem ser executadas para acomodar as mudanças. Estas atividades podem ser as definidas no processo de desenvolvimento ou podem ser mais detalhadas, incluindo tarefas específicas para a implementação destas mudanças.]

- **Escopo**: esforço necessário para atividades e artefatos novos ou afetados (re-trabalho). O esforço pode ser descrito em pontos por caso de uso ou horas de engenharia.
- **Prazo**: preenchido pelo gerente de projeto com a data limite estimada para a conclusão da mudança
- **Custo**: preenchido pelo gerente de projeto com a o custo decorrente do escopo informado.
- **Outro**: qualquer outra descrição de esforço apropriada.

	Atividade	Impactos			
		Escopo	Prazo	Custo	Outro
1					
2					
3					

Figura 4.18: Solicitação de Mudança – Análise de Impacto

No caso de mudanças em requisitos, o impacto no escopo deve ser estimado pelo analista de sistemas. Já as estimativas de prazo e custo devem ser feitas pelo gerente de projeto, com base no impacto em esforço.

Para algumas solicitações de mudança, este passo pode ser executado mais de uma vez, aumentando as informações sobre a análise de impacto a cada execução. Isso acontece com freqüência quando as mudanças são muito grandes ou estruturais. Nestes casos, pode ser interessante fazer uma estimativa inicial menos precisa e executar a próxima atividade do fluxo – *Analisar Solicitação de Mudança*. Assim, o comitê de controle de mudanças pode descartar a mudança mesmo antes de uma análise de impacto mais detalhada, que consumiria recursos do projeto em uma solicitação cujo futuro é a rejeição.

A data de cada análise de impacto deve ser registrada no histórico do documento. Estas datas são importantes porque o impacto vai aumentando ao longo do projeto, quando mais artefatos ficam prontos e precisam acomodar a mudança.

4.6.3 Analisar Solicitação de Mudança

4.6.3.1 Objetivos, Papéis e Artefatos

Esta atividade parte de uma Solicitação de Mudança cuja descrição já foi revista e complementada e cuja análise de impacto já foi documentada (mesmo que apenas em uma estimativa inicial, conforme exemplificado na seção anterior).

O objetivo, neste ponto, é decidir se as mudanças devem ser aprovadas. Os responsáveis por esta decisão são os integrantes do Comitê de Controle de Mudanças. Este comitê é composto, geralmente, por um gerente da organização cliente e o gerente de projeto (equipe de desenvolvimento).

O comitê utiliza as informações documentadas para decidir sobre a adoção ou não da mudança. Se a análise de impacto foi superficial e o comitê ainda acredita que as mudanças podem ser necessárias, a atividade anterior pode ser executada mais uma vez, refinando as estimativas de impacto.

Se o comitê decidir que as mudanças devem ser rejeitadas, esta decisão é documentada na Solicitação de Mudança e a atividade é encerrada. Caso as mudanças sejam aprovadas, a solicitação é atualizada com o novo status e é encaminhada ao gerente de projeto para replanejamento. O re-planejamento, que não faz parte desta atividade, inclui nos planos do projeto as atividades necessárias para acomodar as mudanças.

4.6.4 Resumo do Fluxo de Trabalho

Tabela 4.5: Gerenciar Mudanças – Resumo

Fluxo de Trabalho: Gerenciar Mudanças
Fase de Execução: Iniciação, Elaboração, Construção
Responsáveis: Participantes internos e externos, Comitê de Controle de Mudança, Gerente de Projeto
Artefatos: • Artefatos sobre requisitos (podem ser modificados) • Solicitação de Mudança
Atividades:

• Submeter solicitação de mudança
• Complementar solicitação de mudança
  - Complementar a descrição da mudança
  - Analisar o impacto da mudança
• Analisar solicitação de mudança

Critérios de conclusão:
• Mudança aprovada ou rejeitada.

4.7 Orientações sobre Práticas Ágeis

4.7.1 Análise das Possibilidades de Utilização

Os métodos ágeis foram estudados, neste trabalho, com o objetivo principal de aumentar a produtividade durante a execução das atividades. Após uma ampla revisão bibliográfica, chegou-se às conclusões comentadas a seguir.

Embora diversos métodos ágeis tenham sido propostos nos últimos anos, formando um catálogo variado de técnicas, procedimentos e dinâmicas de desenvolvimento, esses recursos se mostraram de difícil inserção no processo de engenharia de requisitos proposto neste trabalho. Acredita-se que as razões para isso sejam:

1. Os métodos ágeis, por possuírem definições simplificadas e promoverem a auto-organização das equipes de projeto, não possuem um conjunto considerável de elementos de processo voltados para a engenharia de requisitos. A seção *Métodos Ágeis e a Engenharia de Requisitos*, no capítulo 3, resume as estratégias ágeis de desenvolvimento encontradas que se relacionam diretamente à engenharia de requisitos. Como estas estratégias são bastante reduzidas, é difícil identificar sua aplicação ao longo do processo proposto.
2. Os métodos ágeis oferecem os maiores ganhos quando suas práticas são utilizadas em sinergia. Eleger apenas algumas práticas para utilização em um processo de desenvolvimento de *software* é, além de menos produtivo, mais arriscado, já que diversas técnicas só podem ser bem sucedidas quando aplicadas em conjunto com as demais (FOWLER, 2004). Como o processo proposto neste trabalho é restrito à engenharia de requisitos, não se pode inferir que as técnicas utilizadas nas demais disciplinas de desenvolvimento de *software* seguem os métodos ágeis. Este contexto dificulta a inserção de técnicas utilizadas por métodos ágeis no processo proposto, já que a maioria destas técnicas tem a premissa de utilização em conjunto com outros recursos dos métodos ágeis como o jogo do planejamento, a projeto (*design*) evolucionário, a integração contínua e a adoção de testes automatizados.
3. Este trabalho foi realizado em uma organização específica cujos projetos típicos não possuem as características mais indicadas para a aplicação de métodos ágeis. Algumas das características dos projetos desenvolvidos nesta organização são: grandes equipes, compostas de forma heterogênea; clientes internacionais, sendo algumas reuniões de análise realizadas na organização cliente e os contatos seguintes acontecendo por telefone, e-mail ou tele-conferência; ambiente de negócios dinâmico, com uma taxa razoável de mudanças nos requisitos; necessidade de uma certa formalidade na documentação, requisitada pelos clientes. As características citadas não promovem a utilização de métodos ágeis, à exceção da instabilidade dos requisitos.

Considerando as conclusões listadas acima, mas ainda reconhecendo os benefícios que podem ser alcançados com a utilização de métodos ágeis, adotou-se uma estratégia para a inserção desses métodos no processo de engenharia de requisitos proposto. Ao invés de adotar técnicas específicas,

partiu-se para uma abordagem baseada em valores, princípios e práticas dos métodos ágeis que podem ser aproveitados nas atividades do processo proposto. Esta abordagem foi incluída no segmento de melhores práticas do processo proposto. A seção seguinte comenta a abordagem essa abordagem, identificando algumas atividades que podem beneficiar-se de valores, princípios ou práticas dos métodos ágeis.

4.7.2 Propostas de Inserção de Práticas Ágeis no Processo Proposto

As sessões de análise de requisitos em grupo, parte da atividade de Assegurar uma Visão Comum, podem beneficiar-se das práticas de *comunicação face-a-face*, *valorização da simplicidade* e *modelagem com os outros*. A comunicação face-a-face e a modelagem com os outros aplicam-se sobretudo ao passo de análise de requisitos em grupo. A valorização da simplicidade contribui com o processo no passo de revisão de artefatos. Durante as revisões, deve-se tentar perceber qual o mínimo de investimento necessário para que os artefatos relacionados a requisitos contenham informações o suficiente para representar corretamente as necessidades do cliente e para permitir que a equipe técnica implemente o requisito. Nestas reuniões, geralmente ficam claros fatores como: o perfil do cliente e sua capacidade de abstração, o nível de clareza e complexidade de cada módulo do sistema (quando mais simples e claro, menos informações são necessárias no documento), a compreensão da equipe sobre o negócio.

Embora indicadas para a atividade Assegurar uma Visão Comum, as práticas acima aplicam-se a todas as atividades do processo que envolvam desenvolvimento de requisitos.

Outra prática que pode ser utilizada amplamente é adotar *times auto-organizados*. Embora o processo proposto defina as atividades que devem ser executadas, a equipe pode definir seu modo de trabalho e forma de interação, pode subdividir-se para cuidar de mais de um assunto em paralelo ou realizar *workshops* com todos os envolvidos, e pode adotar as técnicas de elicitação e análise que julgar mais produtiva. Algumas estratégias escolhidas pela equipe podem necessitas uma coordenação de atividades com a gerência do projeto.

Uma prática que já é recomendada por RUP é a participação ativa dos *stakeholders*. Em todas as atividades de interação com o cliente, que são estimuladas pela participação ativa, as práticas da *conversação* e *utilização de ferramentas simples* também trazem benefícios. No momento do detalhamento de um requisito, deve-se envolver o cliente, revisando o escopo inicial definido e utilizando as técnicas de elicitação de requisitos descritas na seção de *Melhores Práticas*. Caso o cliente não possa estar fisicamente presente (se estiver, por exemplo, em outro país), a conversação pode ser feita por telefone ou aplicação de mensagem instantânea. O uso de *e-mail* deve ser evitado, pois este meio não permite rápido *feedback* e desacelera a comunicação.

De um modo geral, todas as práticas da Modelagem Ágil podem ser utilizadas nos fluxos de atividades *Definir o Escopo do Sistema* e *Refinar Requisitos de Software*, além da atividade *Assegurar uma Visão Comum*. Destacadamente, as práticas *criar vários modelos em paralelo* e *iterar para outro artefato* aumentam a produtividade e a qualidade do passo Refinar Modelos de Análise.

Por fim, no que diz respeito à Gerência de Mudanças, a prática *receba bem as mudanças* deve ser empregada para evitar a falta de flexibilidade observada na postura das equipes de desenvolvimento de *software* em geral. Qualquer membro da equipe de desenvolvimento ou da organização cliente pode enviar uma solicitação de mudança em qualquer fase do projeto. A decisão sobre discutir a mudança e implementá-la é feita pelo Comitê de Controle de Mudanças, que representa todos os interessados.

A dinâmica da análise de mudanças é avaliar o impacto e envolver o cliente na decisão sobre adotar ou não a mudança. Se a mudança for adotada, o escopo deve ser ajustado, podendo inclusive excluir requisitos de baixa prioridade. Como a decisão é do cliente, não é preciso rejeitar ou mesmo dificultar a adoção da mudança. Esta é uma aplicação da prática *maximizar o investimento dos stakeholders*.

5 INSTITUCIONALIZAÇÃO DO PROCESSO

Conforme foi mencionado no capítulo anterior, o processo de engenharia de requisitos proposto neste trabalho se insere em um processo global de desenvolvimento elaborado por uma organização específica com o objetivo de obter melhorias significativas em produtividade e qualidade de seus produtos de *software*.

Durante cerca de 10 meses, o processo global foi concebido, documentado, divulgado, o ambiente de apoio foi preparado, projetos piloto foram selecionados, membros da equipe receberam treinamento e os primeiros resultados foram colhidos e analisados.

Após os primeiros 3 meses de trabalho, a grande maioria dos projetos desenvolvidos dentro da organização passou a adotar o processo de engenharia de requisitos aqui proposto, embora apenas alguns destes projetos sejam considerados piloto.

As próximas seções descrevem a adoção do processo proposto, incluindo seu planejamento e execução, sempre mantendo o foco na engenharia de requisitos.

5.1 Contexto de Aplicação do Processo

O processo de desenvolvimento proposto foi concebido e implantado dentro de um contexto de aplicação caracterizado por:

1. Uma organização específica;
2. As equipes dedicadas aos projetos piloto;
3. Os clientes atendidos pelos projetos piloto.

Assim, as definições que compõem o processo e seus resultados observados estão intimamente relacionadas às necessidades e possibilidades deste contexto de aplicação.

As seções a seguir descrevem os perfis da organização, equipe e clientes envolvidos nos projetos piloto.

5.1.1 Perfil da Organização

A organização em cujo cenário este trabalho foi institucionalizado está em funcionamento há 13 anos, conta com cerca de 180 colaboradores e focaliza sua atuação em tecnologia de *software* em bancos de dados e web, aplicando a experiência adquirida nas áreas de análise de sistemas, modelagem, projeto e implantação de aplicações corporativas em sua área de atuação.

As tecnologias envolvidas nos projetos desta organização incluem, entre outras:

- *Bancos de Dados*: Microsoft SQL Server, Oracle, DB2, Open Source;
- *Linguagens de Programação*: Java, JSP, EJB, .Net, ASP, PHP, C/C++, Visual Basic;
- *Servidores WEB / Aplicações*: Microsoft MTS, WebSphere, Tuxedo, Oracle iAS, WebLogic, BEA/Jolt;
- *Arquiteturas*: .Net, J2EE.

Os serviços prestados e soluções fornecidas incluem projetos no modelo de fábrica de software e de fábrica de projetos (projetos customizados de acordo com as necessidades dos clientes), iniciativas de melhoria de processos de software, terceirização da equipe de testes de software, suporte técnico e *outsourcing*. Também são oferecidos *mentoring,* consultoria e treinamento em metodologias e técnicas de engenharia de software, arquitetura orientada a objetos, verificação contínua de qualidade, modelagem visual com UML, gerência de requisitos e técnicas de estimativas, automação de testes.

Outro nicho de interesse desta organização é o fornecimento de metodologias e ferramentas de desenvolvimento de software orientadas ao aumento da produtividade e da qualidade da área de Tecnologia de Informação de seus clientes.

5.1.2 Perfil de Clientes e Projetos

O portfólio de clientes da organização em foco reúne empresas de médio e grande porte, de tecnologia, governo ou varejo, em sua maioria contando com um ou mais setores de tecnologia da informação. Grande parte destas empresas situa-se na cidade de Porto Alegre, no estado do Rio Grande do Sul; o restante dos clientes está distribuído ao longo do território nacional e alguns representam clientes internacionais.

No caso específico dos projetos piloto deste estudo, os seguintes perfis clientes estiveram ou estão envolvidos com sua implantação:

- Instituição do governo do estado;
- Fábrica de software e de projetos de uma corporação multinacional com sede em Portugal;
- Projetos internos da organização.

É importante notar que todos os clientes cujos projetos foram considerados para este estudo possuem perfil bastante participativo e puderam contribuir consideravelmente para a definição de escopo e controle de requisitos dos projetos. A gerência e o desenvolvimento de requisitos no processo proposto pressupõem considerável dedicação por parte dos fornecedores e revisores de requisitos da equipe cliente. Caso este envolvimento não possa ser estabelecido, o projeto pode incorrer em atrasos (por demora nas aprovações e revisões) ou na entrega de um produto que não satisfaça às necessidades dos usuários (caso os revisores tenham aprovado as especificações sem dedicar a necessária atenção ao seu conteúdo).

5.1.3 Perfil das Equipes dos Projetos Piloto

As equipes de colaboradores responsáveis pela execução dos projetos piloto seguiram, basicamente, os três perfis abaixo:

- Equipe Pequena: equipes dos projetos internos, compostas por um gerente de projeto, e mais um ou dois integrantes com perfil

multidisciplinar, responsáveis pela análise, projeto, implementação e entrega da solução. Nestas equipes, algumas vezes empregou-se um analista exclusivo para a definição dos requisitos. Enquanto o gerente e o analista possuíam maior senioridade, os demais recursos eram menos experientes, em geral versados em uma das tecnologias empregadas, adquirindo parte do conhecimento ao longo do projeto.

- Equipe Média: equipe do projeto para a Secretaria da Fazenda, composta por um gerente de projeto em tempo integral, um analista (com prévia atuação em tecnologia), um projetista e desenvolvedor sênior e dois desenvolvedores iniciantes.
- Equipe Grande: equipe do projeto da fábrica, composta por um gerente de projeto (com apoio de 2 outros recursos), quatro analistas de sistemas, 2 projetistas em média (chegando a 4 em determinado pico de desenvolvimento) e 5 desenvolvedores em média (tendo chegado a 15). Cerca de 50% da equipe era bastante experiente, e os demais em sua maioria iniciantes.

Observa-se que a composição das equipes envolvidas na implantação do processo foi bastante variada, e todos estes projetos executaram o processo proposto, contribuindo para seu refinamento e para sua avaliação.

5.2 Planejamento da Institucionalização

A adoção do processo de desenvolvimento foi motivada por uma decisão estratégica da diretoria da organização, que deseja investir na melhoria de seus processos. A partir desta decisão, decidiu-se conceber e implantar um processo de desenvolvimento de software na organização que tivesse as seguintes características:

- Ser baseado no Processo Unificado da Rational (RUP), já que este processo já orientava as atividades das equipes de desenvolvimentos da empresa.
- Satisfazer as metas do CMMI, uma vez que a compatibilidade com um processo de qualidade internacionalmente reconhecido aumenta os atrativos para potenciais clientes nacionais e internacionais. O processo deveria atingir maturidade de nível 2 do CMMI. Mais tarde, decidiu-se evoluir apenas o processo de engenharia de requisitos para atingir o nível 3 de maturidade.
- Primar pela praticidade e eficiência, evitando atividades burocráticas e geração de artefatos que não ofereçam índices interessantes de retorno do investimento.
- Aproveitar conhecimentos adquiridos durante atividades de consultoria. na área de processos de desenvolvimento, a uma empresa cliente cujo processo adotado obteve certificação no nível 2 de maturidade do CMMI.

5.2.1 Grupo de Processo de Software (GPS)

O próximo passo no planejamento foi definir uma equipe responsável pela definição e institucionalização do processo. Cada membro da equipe é especialista em uma ou mais áreas de processo:

1. Requisitos (Gerência e Desenvolvimento de Requisitos);

2. Gerência (Planejamento de Projeto, Controle e Monitoração de Projeto);
3. Qualidade e métricas (Garantia de Qualidade de Processo e Produto, Medição e Análise);
4. Controle de versões (Gerência de Configuração);
5. Testes (Validação e Verificação).

Conforme mencionado anteriormente, o processo global foi definido com o objetivo de satisfazer as metas do nível 2 de maturidade do CMMI. Dentre as áreas de processo citadas acima, as únicas que não fazem parte das metas deste nível são: Desenvolvimento de Requisitos, Validação e Verificação. A razão para que estas áreas, que fazem parte do nível 3 de maturidade, fossem incluídas no planejamento, era o fato de a organização já possuir um processo razoavelmente definido e eficiente para execução de atividades relacionadas a tais áreas.

Além dos 5 especialistas, em regime parcial de dedicação, o GPS também conta com um participante de formação mais geral, e com conhecimentos específicos sobre todas as áreas de processo, metas e práticas do CMMI. Por fim, um membro sênior da diretoria atua como patrocinador e padrinho do projeto de institucionalização do processo de desenvolvimento.

5.2.2 Projeto de Institucionalização do Processo

5.2.2.1 Definição do Projeto Interno

O especialista em gerência de projeto foi o coordenador do esforço de definição e institucionalização do processo global de desenvolvimento na organização. Este coordenador criou um projeto interno que representa este esforço, ou seja, a própria institucionalização do processo de desenvolvimento é parte de um projeto interno da organização.

Atualmente, este projeto já atingiu os marcos de *definição do processo* e *institucionalização do processo*, e segue em busca de dois outros marcos: *revisão de resultados* e *implantação de melhorias*. O quinto marco que fazia parte do planejamento inicial era a *avaliação*. Esta foi adiada em função do cancelamento de alguns projetos piloto cujos resultados seriam necessários para a realização da avaliação.

5.2.2.2 Planejamento, Controle e Monitoração

O primeiro projeto a utilizar o processo de desenvolvimento global foi o próprio projeto interno de institucionalização deste processo. Como o processo estava sendo definido ao longo do projeto, este não serviu como projeto piloto. Contudo, o projeto interno possibilitou a percepção de como seria a adoção do processo, a identificação de dificuldades, o retorno inicial por parte da equipe e as primeiras lições aprendidas.

O planejamento, o controle e a monitoração do projeto foram feitos pelo especialista em gerência de projetos, seguindo as orientações definidas no processo. Assim, a definição do cronograma, marcos de projeto, alocação de recursos e demais decisões de projeto ficaram a cargo deste especialista. Estas atividades, assim como o conhecimento gerado em decorrência de sua execução, não fazem parte do escopo desta dissertação.

5.2.2.3 Engenharia de Requisitos no Projeto Interno

O processo de engenharia de requisitos descrito neste trabalho (ver capítulo *Processo Proposto*) insere-se no processo global de desenvolvimento mencionado nas seções anteriores, e no projeto interno citado.

A versão inicial do processo de engenharia proposto contemplava apenas a área de processo de Gerência de Requisitos, e não incluía a recomendação de práticas ágeis. Esta versão inicial foi implantada e executada em alguns projetos piloto. A versão final do mesmo, que é a apresentada neste trabalho, inclui a área de processo de Desenvolvimento de Requisitos e possui orientações para a adoção de valores, princípios e práticas ágeis .ao longo do processo.

5.2.2.4 Áreas de Trabalho

O projeto de institucionalização do processo global de desenvolvimento foi planejado de forma a executar uma série de atividades, agrupadas em áreas de trabalho. Neste contexto, as áreas de trabalho não possuem nenhuma relação com áreas de processo do CMMI; são orientadas ao conceito de *Estruturas de Organização do Trabalho* (WBS – *Work Breakdown Structure*) de projetos.

As áreas de trabalho de atividades planejadas foram:

- definição do processo;
- documentação do processo;
- preparação *cultural* da organização;
- treinamento de pessoal e *mentoring*;
- execução do processo;
- avaliação e melhoria do processo.

Diversas atividades destas áreas de trabalho foram executadas ao longo do projeto e algumas continuam em execução. As seções seguintes descrevem as áreas de trabalho sob o enfoque das atividades relevantes para este trabalho, ou seja, atividades relacionadas à engenharia de requisitos.

5.3 Áreas de Trabalho

5.3.1.1 Definição e Documentação do Processo

Para a definição do processo proposto, foi necessário o estudo de sua fundamentação teórica, ou seja, o estudo das abordagens CMMI, RUP e Métodos Ágeis. Este estudo ocorreu durante 22 meses, de março de 2004 a dezembro de 2005, e continua em execução. As atividades desenvolvidas ao longo deste período foram:

- Assistir a aulas de pós-graduação relacionadas ao tema;
- Receber treinamento oficial do SEI/CMU sobre CMMI;
- Realizar revisão bibliográfica sobre estas abordagens e possibilidades de integração das mesmas;
- Complementar a revisão bibliográfica acompanhando e participando de discussões sobre estes temas em reuniões, congressos e listas de discussão;

- Consolidar os estudos em métodos ágeis elaborando um trabalho individual de pesquisa acadêmico;
- Consolidar os estudos em métodos ágeis elaborando um trabalho individual de pesquisa acadêmico;
- Participar de um projeto em uma organização certificada em CMMI, no nível de maturidade 2;
- Consolidar os estudos sobre RUP criando e ministrando um curso comercial sobre engenharia de requisitos e planejamento de projetos com UML e o Processo Unificado.

A partir de fevereiro de 2005, o grupo GPS passou a realizar reuniões semanais de andamento e reuniões semanais de revisão do processo proposto. As definições do processo para as áreas de processo de Gerência de Requisitos e de Desenvolvimento de Requisitos foram sendo gradualmente concebidas e documentadas, e então validadas nas reuniões semanais.

As iterações de pesquisa, definição, implantação, avaliação e ajuste do processo, foram descritas anteriormente, no capítulo 3, *Processo de Engenharia de Requisitos Proposto*.

As orientações sobre o processo foram disponibilizadas aos membros da organização, em diferentes momentos, de acordo com o que será descrito nas seções seguintes, sobre preparação da organização, treinamento e apoio ao processo. As principais informações sobre a documentação do processo, suas orientações e a influências das abordagens que o fundamentam estão reproduzidas no capítulo 3, sobre o *Processo de Engenharia de Requisitos Proposto*.

5.3.2 Preparação “Cultural” da Organização

Diversos estudos evidenciam dificuldades encontradas na implantação de um (novo) processo de desenvolvimento de *software* em uma organização. A resistência oferecida pela cultura organizacional é identificada como uma das principais dificuldades nesta implantação.

A ligação da resistência de organizações à adoção de processos de desenvolvimento baseados em CMM ou CMMI ao fator cultural é descrita em trabalhos como (MCCOY; MARTIN, 2005), (DUTTON; SVERDRUP, 2002), (WIEGERS, 2005), (RIEMENSCHNEIDER; HARDGRAVE; DAVIS, 2002). Estes trabalhos descrevem justificativas para esta resistência e estratégias para superá-la.

Durante a institucionalização do processo de engenharia de requisitos proposto, os esforços feitos no sentido de minimizar a resistência da organização à adoção do processo foram chamados de *preparação cultural da organização*. Estudos mostram que a institucionalização de um processo só pode ser realmente obtida se os membros das equipes de projeto forem *convencidos* a utilizar tal processo. Para isto, não basta uma comunicação, partindo da diretoria ou do Grupo de Processo de Software, que exija a utilização do processo (RIEMENSCHNEIDER; HARDGRAVE; DAVIS, 2002).

Algumas das justificativas, identificadas na literatura, para a resistência à adoção de novos processos de desenvolvimento são:

- A crença de que o modo de trabalho da organização não é compatível com o processo que está sendo implantado;

- A percepção de que a utilização do novo processo envolve uma carga adicional de trabalho cujo benefício não valida o investimento e, por isso, é prejudicial aos clientes;
- O investimento necessário, por parte dos membros da organização, para dedicarem-se ao aprendizado de novas técnicas e atividades;
- Os responsáveis pela gerência e liderança na organização, cujas ações contribuíram para alcançar os objetivos traçados até o momento, precisam adotar as novas orientações de um modelo complexo que ainda está sendo compreendido.

Exemplos de estratégias para diminuir a resistência da organização ao processo são:

- Envolver pessoas chave da organização na definição do processo;
- Utilizar os argumentos corretos para motivação de pessoal, destacando a *utilidade* da metodologia: resolução de problemas enfrentados atualmente na organização, melhorias na qualidade e produtividades, diminuição dos esforços de manutenção voltados para correção de erros.
- Identificar dificuldades da equipe com o processo atual ou com a institucionalização do processo e trabalhar estas dificuldades na definição e melhoria do novo processo;
- Proteger os clientes dos gastos iniciais com a institucionalização do processo, através de investimentos internos no mesmo;
- Efetuar a comunicação e revisão freqüente do processo com os membros da organização e com a gerência sênior;
- Elaborar respostas positivas e pró-ativas às objeções e críticas ao novo processo;
- Alinhar os objetivos e a configuração do processo proposto aos valores estabelecidos na organização.

No contexto do processo de engenharia de requisitos proposto neste trabalho, as seguintes ações foram tomadas para mitigar a resistência cultural da organização ao processo:

- Comunicar, a todos os membros da organização, os objetivos do processo, o andamento de sua definição e institucionalização, os marcos atingidos, as contribuições incorporadas. Algumas ações neste sentido foram: um evento de lançamento do processo, comunicações regulares em meio eletrônico, caracterização do ambiente de trabalho com consciência do processo, trabalhos feitos no sentido de dar apoio ao processo, como promover treinamentos, disponibilizar a descrição do projeto a toda a organização e fornecer mecanismos de sugestão de melhorias (ver seções seguintes: *Treinamento* e *Apoio ao Processo*).
- Criar um processo de engenharia de requisitos baseado em práticas de sucesso já adotadas na organização, baseadas em RUP.
- Delegar, a um analista de sistemas atuante na organização, a tarefa de coordenar a definição do processo de engenharia de requisitos a ser implantado – o analista em questão é a autora deste trabalho.
- Consultar outros analistas de sistemas e demais envolvidos em atividades de engenharia de requisitos para tomar decisões chave

relacionadas à definição do processo; submeter descrições de metas e atividades, modelos de artefatos e demais componentes do processo a estes envolvidos.

- Adotar uma abordagem positiva ao receber críticas, explicando as decisões tomadas sobre a definição do processo com foco no retorno do investimento. É importante observar, com relação a este item, que é inevitável deixar alguns membros da equipe descontentes ou desconfortáveis, o que ocorre na adoção de qualquer processo. Da mesma forma, algumas pessoas costumam burlar os processos definidos de forma a continuarem executando o *seu "próprio processo"*. É importante adotar mecanismos de garantia de qualidade que façam com que estas pessoas utilizem o processo para depois disto darem suas contribuições e sugestões de melhoria.
- Absorver os custos do esforço de definição do processo, registrados sob o projeto interno criado para este fim, sem cobrá-los dos clientes da organização. A crença dos idealizadores do processo, bem como da diretoria da organização, é a de que a utilização do processo, em longo prazo, deve reduzir os custos e aumentar a produtividade das equipes; assim, o custo não precisa ser repassado aos clientes.

A figura a seguir ilustra uma série de eventos promovidos para a execução das ações descritas acima.

Figura 5.1: Eventos realizados para diminuir a resistência cultural ao processo

5.3.3 Treinamento de Pessoal

Para que o processo possa ser utilizado, não basta disponibilizar sua definição para os interessados; é preciso treinar as pessoas que serão responsáveis por executá-lo. De fato, uma das práticas genéricas relacionadas

ao nível de maturidade 2 do CMMI trata justamente do treinamento do pessoal que executa e dá suporte ao processo.

Com relação à engenharia de requisitos, foram criados os seguintes módulos de treinamento, como parte da institucionalização do processo:

- Treinamento em Gerência de Requisitos;
- Treinamento em Desenvolvimento de Requisitos;
- Treinamento em Modelagem de Casos de Uso (este treinamento passou a integrar o portfólio de treinamentos comerciais oferecidos pela organização);
- Treinamento em Processo Unificado com UML (este treinamento passou a integrar o portfólio de treinamentos comerciais oferecidos pela organização).

Estes treinamentos foram oferecidos a alguns integrantes dos projetos piloto, a alguns membros da organização e a uma organização cliente (treinamentos comerciais). Diversas turmas ainda precisam ser formadas para que todo o pessoal da empresa esteja apto a aplicar o processo definido.

Além dos treinamentos, outra iniciativa contribuiu para a compreensão e adoção do processo pelas equipes de projeto: a realização de *mentoring* das atividades e dos artefatos relacionados à engenharia de requisitos. A prática do *mentoring* consiste na orientação, por indivíduos versados no processo, sobre como proceder nas atividades diárias do projeto. Efetivamente, esta prática faz parte da área de processo de Garantia de Qualidade.

Figura 5.2: Artefatos de sessões de treinamento

5.3.4 Execução do Processo

Uma vez definido o processo e treinadas as equipes, partiu-se para a execução do processo. A escolha de projetos piloto foi feita de acordo com o portfólio de projetos da organização e com as características de cada projeto: tamanho do projeto, capacitação da equipe, cliente envolvido, local de desenvolvimento.

Embora o projeto de institucionalização do processo estivesse monitorando os projetos piloto com grande atenção, os demais projetos desenvolvidos internamente também passaram a utilizar o processo de engenharia de requisitos proposto. Isto facilitou a institucionalização do processo, fez com que as soluções para problemas comuns descritas nos projetos pudessem ser aproveitadas por todos os interessados e permitiu aumento do *feedback* recebido sobre o processo definido.

O processo executado é o descrito no capítulo do *Processo de Engenharia de Requisitos Proposto*. Durante sua execução, foram utilizados os recursos de *mentoring*, melhores práticas, repositórios e ferramentas descritos na seção de *Apoio ao Processo*.

5.3.5 Avaliação e Melhoria do Processo

Durante sua definição, o processo foi constantemente reavaliado e complementado, a partir do feedback dos membros da organização e dos projetos piloto.

As modificações, contudo, foram inseridas em pequenos incrementos, de forma controlada e bem definida. A cada mudança, se algum projeto piloto já havia iniciado a execução de uma atividade alterada, este seguia utilizando a versão antiga do processo.

A única modificação drástica foi a inclusão das práticas relacionadas à área de processo de Desenvolvimento de Requisitos, responsável pela inclusão de alguns fluxos de atividades no processo.

5.4 Apoio ao processo

O processo institucionalizado representa a orientação padrão necessária para o planejamento e a execução de projetos na organização. Contudo, uma série de recursos de apoio ao processo está disponível às equipes de desenvolvimento. Estes recursos incluem, além dos treinamentos e atividades de *mentoring*, descritos anteriormente, descrições de melhores práticas de desenvolvimento e gerência de requisitos e suporte automatizado ao processo por meio de repositórios e ferramentas.

5.4.1 Melhores Práticas

As principais descrições disponibilizadas sobre melhores práticas relacionadas à engenharia de requisitos dizem respeito a técnicas de elicitação e descrição de requisitos e as orientações sobre aplicação de práticas ágeis.

As técnicas listadas a seguir fazem parte do *Treinamento em Processo Unificado com UML*:

- *Entrevistas e Questionários*: descrição das etapas a serem seguidas em uma entrevista e como documentá-la e conduzi-la, além e observações sobre a utilização de questionários.
- *Workshop de Requisitos*: este é um evento executado em um curto espaço de tempo, reunindo todos os *stakeholders* com o objetivo de levantar uma versão inicial dos requisitos do sistema. Sua descrição inclui os objetivos, funcionamento, preparação, passos e resultados, além de estratégias a serem utilizadas durante o *workshop*.
- *Brainstorming* e *Redução de Idéias*: são estratégias utilizadas em diversas atividades de análise de requisitos. Representam, respectivamente, um levantamento amplo de idéias e a organização e priorização destas idéias.
- *Prototipação* e *Storyboarding*: são utilizadas com o objetivo de obter conhecimento sobre o funcionamento do sistema antes de sua construção. Sua descrição inclui os tipos de protótipos e as situações que podem beneficiar-se da aplicação destas estratégias.
- *Modelagem de Casos de Uso*: é uma técnica de elicitação e análise de requisitos amplamente utilizada. A modelagem de casos de uso é descrita em detalhes, uma vez que é sugerida em diversas atividades do processo proposto.
- *Workshop de Casos de Uso*: é um tipo específico de *workshop* de requisitos, com etapas definidas como a identificação de atores e casos de uso ou a consolidação do modelo de casos de uso.
- *Diagramas de Atividade*: da forma como está descrito nas melhores práticas do processo, pode ser utilizado para modelagem de negócios (não incluída no processo proposto) ou para detalhamento de requisitos.

A principal fonte de pesquisa consultada para a descrição das práticas acima é o livro *Mananging Software Requirements, A Unified Approach*, cujos ensinamentos são totalmente integrados a RUP (LEFFINGWELL; WIDRIG, 2000). As observações do livro foram complementadas com sub-práticas e artefatos sugeridos pelo CMMI (SOFTWARE ENGINEERING INSTITUTE, 2002) e por experiências específicas da organização.

As orientações sobre a utilização de práticas ágeis estão descritas no capítulo 4, *O Processo de Engenharia de Requisitos Proposto*, na seção *Proposta de Inserção de Práticas Ágeis*.

5.4.2 Automação do Processo: Repositórios e Ferramentas

Outro recurso utilizado para dar apoio à institucionalização do processo proposto é a automação do ambiente de execução do processo. Esta automação foi implementada através de repositórios de informação e ferramentas de auxílio ao desenvolvimento. No contexto da engenharia de requisitos, as iniciativas de automação adotadas foram:

- A criação de um repositório de artefatos da organização, que é alimentado com os artefatos gerados pela equipe de desenvolvimento, ao final da execução de cada projeto realizado;
- A criação de um repositório de artefatos do projeto em desenvolvimento, que é alimentado com os artefatos gerados pela equipe de desenvolvimento, ao longo da execução de cada projeto realizado;

- Utilização da ferramenta IBM/Rational RequisitePro (IBM RATIONAL CORPORATION, 2005b), (RATIONAL UNIVERSITY, 2002) para auxiliar a documentação e a gerência de requisitos.

Os dois repositórios de documentos são implementados utilizando o *IBM/Rational Clear Case* (IBM RATIONAL CORPORATION, 2005), uma ferramenta de controle e gerência de artefatos de software utilizada ao longo do ciclo de vida de projetos de software. Os principais benefícios atingidos como resultado da utilização do *Clear Case* são a viabilização e o controle de acesso e de versão, além do armazenamento seguro dos artefatos.

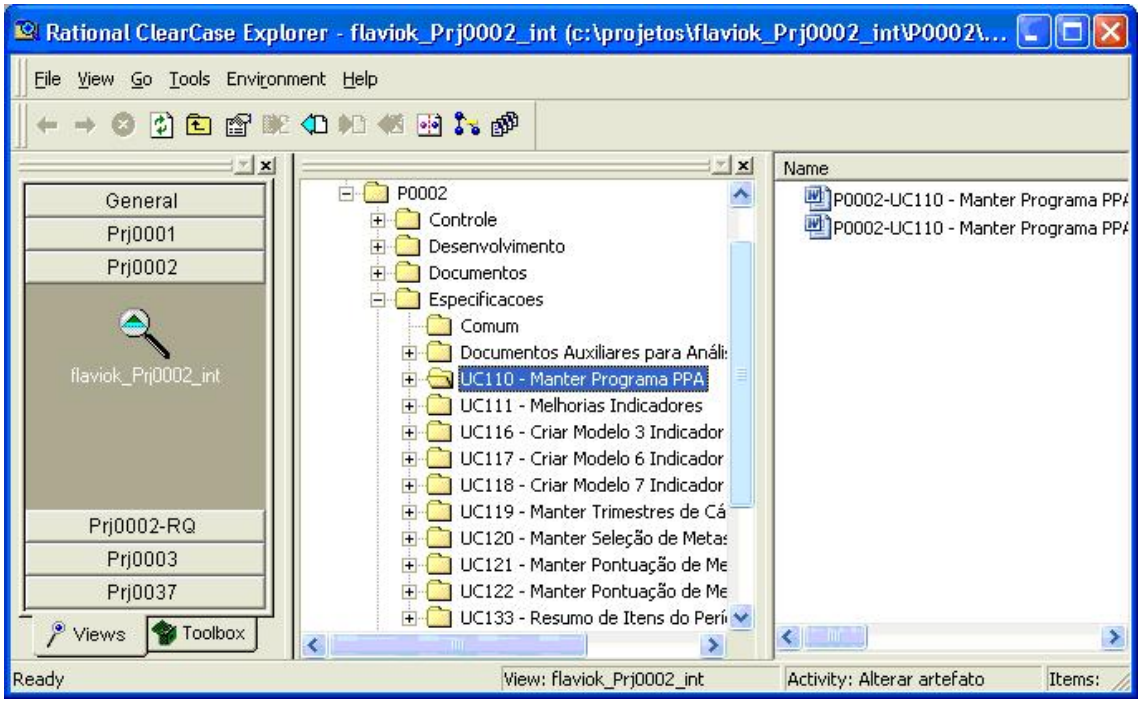


Figura 5.3: Área de Trabalho da ferramenta IBM/Rational Clear Case

A ferramenta IBM/Rational RequisitePro, ou simplesmente *ReqPro*, possibilita o armazenamento e a gerência de requisitos, sua documentação, a gerência de atributos associados e de matrizes de rastreabilidade, o controle do histórico de modificações.

Os seguintes artefatos são armazenados no repositório do ReqPro:

- Documento de Visão
- Especificação de Requisitos de Software
- Especificação Funcional de Requisito
- Especificação Técnica de Requisito

Como a ferramenta é totalmente integrada do *Microsoft Word*, todos estes documentos são criados e editados utilizando-se *MS-Word*. O ReqPro também fornece controle de acesso e de versão. Um exemplo de cenário típico de utilização do ReqPro é a criação do documento de Visão:

1. Um analista de sistemas cria o documento com seu conteúdo inicial e o importa para o repositório do ReqPro.
2. O analista marca o texto correspondente a cada necessidade e característica do sistema, descritas no Visão, e as registra no ReqPro.
3. Salva e fecha o documento.

4. Outro analista abre o documento e insere mais características.
5. Salva e fecha o documento.
6. Um analista verifica a matriz de rastreabilidade de *Necessidades x Características*, inicialmente vazia.
7. O analista preenche a matriz com as dependências entre estes requisitos.
8. Salva e fecha o documento.

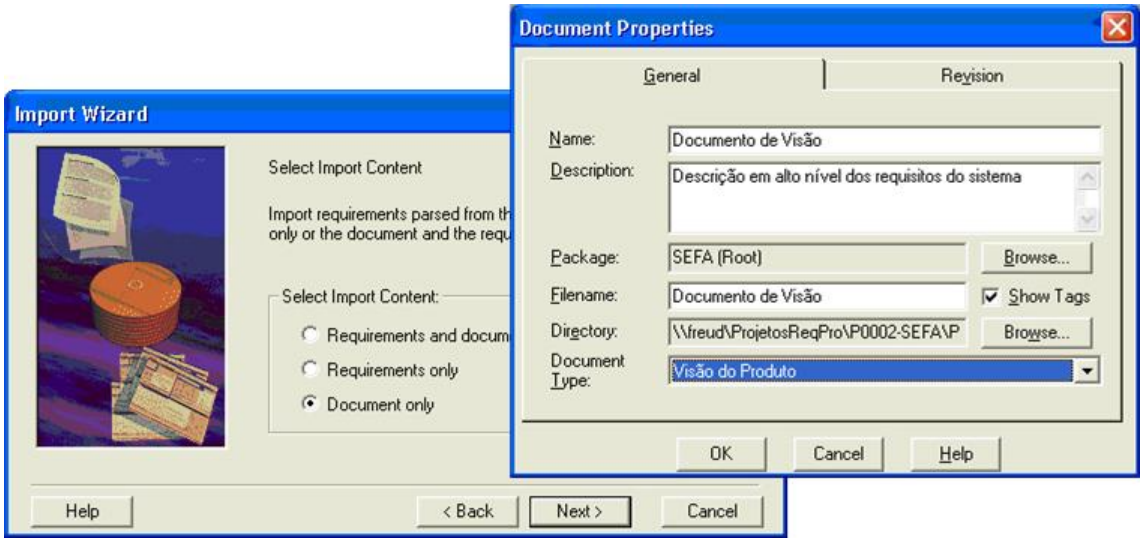


Figura 5.4: Criação do Documento de Visão no RequisitePro

Os tipos de requisitos e as matrizes são configuráveis, e foram criados especialmente para dar suporte ao processo de engenharia de requisitos proposto. A cada projeto, o ambiente de desenvolvimento é instalado pela equipe responsável pela gerência de configuração e, após isto, todos os tipos de requisitos e as matrizes de rastreabilidade ficam disponíveis para uso pela equipe do projeto.

A automação da gerência de requisitos, utilizando o IBM/Rational RequisitePro, permite alta produtividade em tarefas como:

- Notificação de necessidade de revisões em requisitos;
- Análise de impacto de mudanças;
- Recuperação de histórico de requisitos;
- Relatórios de status de requisitos de *software*.

Estas tarefas são descritas nas seções a seguir. Note-se que a maioria delas está relacionada à atividade *Gerenciar Requisitos*, do processo proposto.

5.4.2.1 Notificação de necessidade de revisões em requisitos

Os requisitos são documentados no repositório do ReqPro, juntamente com suas dependências, através das matrizes de rastreabilidade. Como regra geral, um documento só é inserido no repositório após sua aprovação, quando então deve ter suas mudanças analisadas e controladas.

Após este momento, cada vez que um requisito é alterado, todos os requisitos relacionados a ele são identificados como *suspeitos*. Cada requisito suspeito é revisado pelo analista de sistemas, que verifica se a alteração feita no requisito original causa impacto nos demais. Caso o requisito suspeito não sofra impacto, ou após realizar as modificações necessárias para manter o

requisito suspeito consistente com a alteração feita no requisito original, o analista de sistemas *desmarca* o requisito suspeito.

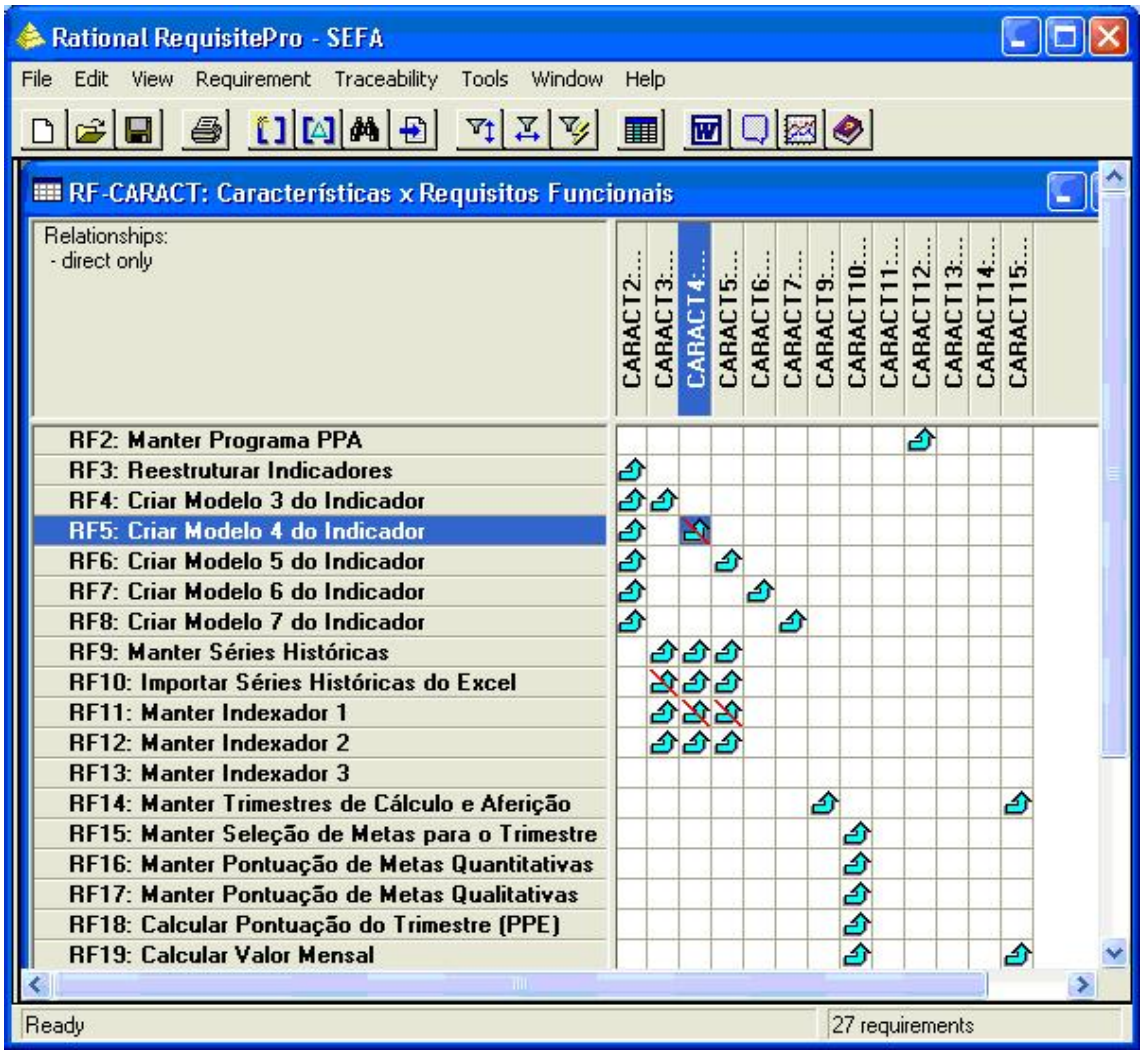


Figura 5.5: Requisitos marcados como *suspeitos*

5.4.2.2 Análise de impacto de mudanças

Esta tarefa também se apóia nas matrizes de rastreabilidade de requisitos. Para analisar o impacto de uma mudança em determinado requisito (necessidade, característica ou requisito de software), verifica-se a árvore de rastreabilidade de requisitos. Esta árvore mostra todos os demais requisitos associados ao requisitos cuja mudança está sendo considerada, e contribui para a estimativa de esforço necessário na adoção da mudança.

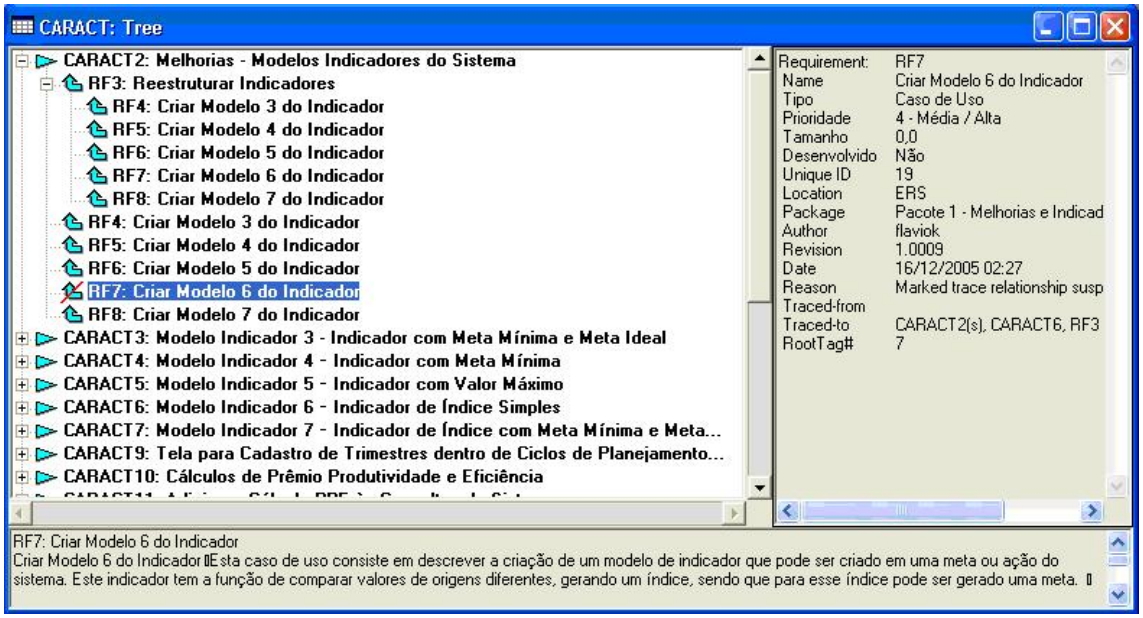


Figura 5.6: Árvore de rastreabilidade de requisitos

5.4.2.3 Recuperação de histórico de requisitos

Depois de inserido no repositório, todo documento sobre requisitos fica sob controle de acesso e de versão. Assim, é possível verificar cada mudança realizada, a data de realização, o analista responsável e inclusive o texto alterado.

Cada vez que uma alteração é realizada, o analista registra o motivo da alteração. Esta é uma fonte importante de informações, já que fornece um mecanismo para relatório de justificativas de alteração dos requisitos, ao longo de todo o projeto.

Figura 5.7: Histórico de alteração de requisitos

5.4.2.4 Relatórios de status de requisitos de software

Cada um dos requisitos (necessidade, característica ou requisitos de software) armazenados no repositório possui atributos associados: status, tamanho, prioridade, data de criação e última alteração, analista responsável, programador responsável, etc. Estes atributos podem ser utilizados nas atividades da área de processo de *Monitoração de Controle de Projeto*, gerando relatórios de andamento do projeto.

5.5 Conformidade do Processo com CMMI

O processo de engenharia de requisitos proposto neste trabalho foi definido com base nas metas e práticas das áreas de processo de Gerência de Requisitos (nível 2 de maturidade) e Desenvolvimento de Requisitos (nível 3 de maturidade) do CMMI. Assim, por definição, está em conformidade com estas duas PAs, o que significa que pode ser integrado a um processo global utilizado por uma organização de nível 3 de maturidade.

Contudo, algumas práticas das PAs consideradas na definição deste processo de engenharia de requisitos dependem de disciplinas do processo de desenvolvimento global diferentes da disciplina de Requisitos. Neste caso, a conformidade com CMMI só pode ser comprovada quando o processo global é analisado, o que não faz parte do escopo deste trabalho.

A tabela a seguir ilustra a conformidade com CMMI alcançada pelo processo proposto, evidenciando as práticas dependentes de outras disciplinas do processo global de desenvolvimento. Para cada prática recomendada pelo CMMI, é indicada a atividade que a adota no processo de engenharia de requisitos proposto. As práticas estão agrupadas por meta, e as metas por área de processo.

Tabela 4.6: Satisfação de metas e práticas das PAs REQM e RD do CMMI

	CMMI: Área de Processo – Meta – Prática		**Atividades do Processo Proposto para satisfação das práticas CMMI**	**Disciplinas Complementares necessárias para a conformidade**
REQM (ML2)	**Gerência de Requisitos (*Requirements Management*)**			
SG1	**Gerenciar Requisitos**			
SP1.1	Compreender os requisitos.		Compreender os requisitos do cliente Compreender os requisitos do produto Assegurar uma visão comum	
SP1.2	Obter comprometimento com os requisitos.		Gerenciar Requisitos Gerenciar Mudanças	Gerência de Projeto (planos de alocação, replanejamento)
SP1.3	Gerenciar mudanças nos requisitos.		Gerenciar Mudanças	Gerência de Projeto (replanejamento) Métricas (criação de base histórica)
SP1.4	Manter rastreabilidade bidirecional dos requisitos.		Gerenciar Requisitos	
SP1.5	Identificar inconsistências entre os resultados do projeto e os requisitos.		Assegurar uma visão comum	Gerência de Projeto (revisão dos planos) Garantia de Qualidade (revisão da consistência entre artefatos)
	CMMI: Área de Processo – Meta – Prática		**Processo Proposto: Atividades**	**Outras disciplinas do Processo**
RD (ML3)	**Desenvolvimento de Requisitos (*Requirements Development*)**			
SG1	**Desenvolver requisitos do cliente**			
SP1.1	Elicitar necessidades.		Compreender Requisitos do Cliente	
SP1.2	Desenvolver os requisitos do cliente.		Compreender Requisitos do Cliente	
SG2	**Desenvolver Requisitos do Produto**			

SP2.1	Estabelecer requisitos de produto e de componentes de produto.		Compreender Requisitos do Produto Especificar Requisito de Software Analisar o Domínio	Solução Técnica (decisões de arquitetura e definição de componentes)
SP2.2	Alocar requisitos de componente de produto.		Compreender Requisitos do Produto Especificar Requisito de Software Analisar o Domínio	Solução Técnica (decisões de arquitetura e definição de componentes)
SP2.3	Identificar requisitos de interface.		Especificar Requisito de Software Modelar Interface	Solução Técnica (decisões de arquitetura e definição de componentes)
SG3	**Analisar e Validar Requisitos**			
SP3.1	Estabelecer conceitos operacionais e cenários.		Especificar Requisito de Software	Solução Técnica (decisões de arquitetura e definição de componentes)
SP3.2	Estabelecer uma definição de funcionalidade requerida.		Especificar Requisito de Software	
SP3.3	Analisar requisitos.		Gerenciar Requisitos Assegurar uma Visão Comum	
SP3.4	Analisar requisitos para alcançar equilíbrio.		Compreender Requisitos do Cliente Assegurar uma Visão Comum	
SP3.5	Validar requisitos com métodos abrangentes.		Assegurar uma Visão Comum	Gerência de Projeto (planejamento de acordo com o ciclo iterativo e incremental RUP)

6 CONCLUSÕES

A seguir, são apresentadas as conclusões obtidas após a finalização deste trabalho. Os principais pontos do trabalho são comentados e as dificuldades encontradas são descritas e analisadas. Por fim, são apresentadas as principais contribuições e as possibilidades de trabalhos futuros.

6.1 Considerações sobre a Definição e Institucionalização do Processo

6.1.1 Trabalho Realizado e Conclusões

Este trabalho teve, como principal objetivo, a definição e institucionalização de um processo de engenharia de requisitos que possibilitasse, a uma organização específica, a obtenção do nível 3 de maturidade de processos, de acordo com a *Integração de Modelos de Capacidade e Maturidade* (CMMI). O processo em questão está inserido em um processo global de desenvolvimento, que também está sendo institucionalizado na organização.

Para a construção do processo de engenharia de requisitos, foi utilizada como base uma metodologia que já era adotada na organização: RUP. A partir do *framework* RUP, foram definidos papéis, atividades e artefatos necessários para alcançar as metas e executar as práticas definidas no CMMI. Estas definições foram documentadas no *Processo de Engenharia de Requisitos Proposto*, um processo definido à semelhança de RUP porém voltado para as orientações do CMMI.

O processo proposto possui três fluxos de atividades: Definir o Escopo do Sistema, responsável pela identificação dos requisitos do sistema; Refinar Requisitos de Software, que fornece o detalhamento necessário destes requisitos; e Gerenciar Mudanças, que trata de alterações nos requisitos do sistema a ser desenvolvido. Cada um destes fluxos é composto por atividades que, executadas por membros da equipe ao desempenhar algum papel específico, recebem e geram artefatos.

Ao longo da definição do processo proposto, procurou-se identificar pontos de inserção de práticas descritas em métodos ágeis. O benefício buscado com a utilização destas práticas foi um aumento na produtividade da engenharia de requisitos. Percebeu-se uma série de dificuldades na utilização destas práticas, devido à falta de foco destas práticas em atividades de engenharia de requisitos, a dificuldade no emprego de práticas isoladas e as características de projetos típicos da organização, não indicadas para métodos ágeis. Ainda assim, os valores, princípios e práticas ágeis foram documentados e um conjunto de orientações para a utilização destes conceitos nas atividades do

processo de engenharia de requisitos proposto foi disponibilizado no segmento de melhores práticas do processo.

A institucionalização do processo proposto na organização foi iniciada após a conclusão da primeira versão do processo, e continua em execução. As principais atividades incluídas nesta institucionalização são a preparação cultural da empresa para evitar resistências ao processo, o treinamento e *mentoring* das equipes de projeto, a execução do processo e sua avaliação e inclusão de melhorias.

Existe uma série de recursos de apoio ao processo: são documentadas melhores práticas de desenvolvimento e gerência de requisitos; existe uma orientação sobre práticas ágeis adequadas a cada atividade ou artefato do processo; e o ambiente de desenvolvimento possui ferramentas de auxílio à engenharia de requisitos, que estão configuradas de acordo com as necessidades do processo proposto.

Durante a institucionalização do processo, percebeu-se a importância da experimentação e avaliação constantes dos componentes de processo definidos. Algumas atividades ou artefatos parecem atender perfeitamente às necessidades da organização ao serem definidos; contudo, quando uma equipe de projeto tenta utilizá-los, problemas de adequação ou de produtividade são percebidos.

A próxima seção descreve algumas dificuldades encontradas ao longo da realização deste trabalho.

6.1.2 Dificuldades Encontradas

Uma série de dificuldades foi identificada durante a definição e institucionalização do processo de engenharia de requisitos proposto. Estas dificuldades têm origem tanto em problemas genéricos, enfrentados por diversos esforços de melhoria de processo, como também em problemas específicos da organização onde este trabalho foi aplicado, e seu portfólio de projetos atual.

O estudo da fundamentação teórica, revisões bibliográficas, treinamentos e participação em discussões sobre as abordagens de RUP, CMMI e melhorias de processos de *software* ocorreu com tranqüilidade. Porém, desde o início da concepção do processo de engenharia de requisitos proposto, ficou evidente a necessidade de recursos consideráveis para as demais atividades do projeto de institucionalização do processo.

Foi necessário alocar tempo para as atividades de definição, planejamento, documentação e execução do processo, além de *mentoring*, análise de resultados e refinamento do processo. Além disso, *software* e *hardware* precisaram ser adquiridos para a implantação do ambiente de auxílio ao desenvolvimento. As ferramentas IBM/Rational ClearCase e RequisitePro já estavam sendo utilizadas amplamente na organização. Porém, com a criação de repositórios únicos e configurações padrão, foi necessário adquirir mais licenças de *software* e realizar *upgrades* nas máquinas que contêm os servidores destas ferramentas.

Outro problema percebido foi uma tendência à incorporação, no processo proposto, de atividades não essenciais para alcançar as metas do CMMI. Uma possível justificativa para este comportamento é a ampla gama de metodologias, modelos de processo, métodos, técnicas e melhores práticas

disponíveis na literatura. Ao estudar estes conceitos, o projetista de processos tende a sobrecarregar o processo definido com a intenção de utilizar os recursos disponíveis para análise e gerência de requisitos, mitigação de riscos, garantia de qualidade, aumento de produtividade, entre outros benefícios. Contudo, inserir uma sobrecarga de atividades e artefatos no processo padrão de engenharia de requisitos acaba diminuindo a produtividade e os ganhos em qualidade, dificultando a execução de planos, e pode resultar em um processo burocrático e de difícil implantação.

O problema de sobrecarga do processo com atividades não essenciais foi resolvido com a adoção de uma postura minimalista, seguindo com uma das características básicas que orientaram a criação do processo: *primar pela praticidade e eficiência*. As metodologias, enfoques e melhores práticas da literatura julgadas importantes para a engenharia de requisitos na organização foram documentadas como recursos e recomendações, fora do escopo de definição do processo de desenvolvimento global.

Uma dificuldade pertinente a este trabalho foi a necessidade de isolar a documentação das atividades de engenharia de requisitos das outras áreas de processo. Durante a definição do processo, os especialistas responsáveis por cada área de trabalho interagiram intensamente. Na própria definição, em forma de páginas HTML disponíveis na rede interna (*intranet*) da organização, cada atividade de engenharia de requisitos possui ligações (*links*) com atividades das outras áreas de processo. Contudo, neste trabalho, como as outras áreas de processo não estão completamente descritas, não ficam claramente evidenciadas as ligações com áreas de processo como o *Planejamento de Projeto*, *Monitoração e Controle de Projetos*, *Gerência de Configuração*, entre outras.

O isolamento descrito acima impede a garantia de conformidade com nível 3 de maturidade do CMMI, já que existem algumas dependências entre as práticas das áreas de processo de Gerência de Requisitos e Desenvolvimento de Requisitos e as demais áreas de processo do CMMI. Também dificulta a adoção de práticas de métodos ágeis, já que estas práticas são bastante dependentes das demais, voltadas para o planejamento e a solução técnica.

Por fim, um problema específico da organização onde este trabalho foi desenvolvido diz respeito aos projetos piloto e, conseqüentemente, ao levantamento de resultados relacionados à adoção do processo. Em função de demandas de mercado, o perfil de projetos da organização teve uma mudança considerável: o número de projetos internos, até então apenas um pouco menor que o de projetos externos, ficou extremamente reduzido. Projetos internos são aqueles desenvolvidos e gerenciados *in house*, seguindo o processo de desenvolvimento da organização. Já os projetos externos são aqueles desenvolvidos em clientes, sob a gerência de pessoal externo à organização e assim sujeitos aos processos de desenvolvimento adotados pelas organizações clientes.

Com isso, o planejamento e controle do projeto interno de institucionalização do processo de desenvolvimento global enfrentou problemas na definição de projetos piloto, e a visibilidade dos resultados obtidos ficou comprometida.

Além disto, a área de processo de Análise e Medição ainda não foi completamente institucionalizada, o que também dificulta a obtenção de

resultados objetivos sobre a aplicação do processo. Sob o ponto de vista da engenharia de requisitos, estes resultados não estão consolidados de maneira satisfatória para inclusão neste trabalho.

Apesar das dificuldades, o processo de engenharia de requisitos proposto foi institucionalizado, está sendo utilizado pelos projetos internos da organização e continua em constante avaliação e melhoria. As contribuições decorrentes deste trabalho são descritas a seguir.

6.2 Contribuições do Trabalho

A principal contribuição deste trabalho é a descrição de um estudo de caso de unificação de abordagens amplamente utilizadas atualmente na definição de processos de desenvolvimento de software: CMMI com a definição de metas e práticas; RUP como base para a definição de fluxos de atividades, papéis e artefatos; e métodos ágeis como boas práticas sugeridas na execução do processo. Este estudo de caso pode ser utilizado por organizações cujo processo de desenvolvimento está em fase de definição ou de reformulação para: compreender as alternativas de integração das abordagens utilizadas; obter conhecimento sobre as áreas propensas a gerar dificuldade na definição e implantação de um processo de desenvolvimento, especificamente um processo baseado nas abordagens de CMMI, RUP e métodos ágeis; compreender como metas, práticas e princípios de abordagens distintas de desenvolvimento de *software* podem contribuir para a melhoria do processo adotado pela organização.

O processo de engenharia de requisitos proposto também é uma contribuição deste trabalho. Este processo está descrito em detalhes e pode ser utilizado por organizações interessadas na melhoria de seus processos e resultados.

Os capítulos iniciais, de fundamentação, podem ser utilizados por leitores interessados em obter uma visão geral das abordagens discutidas (CMMI, RUP e métodos ágeis) e, principalmente, como uma discussão de possibilidades de implantação destas abordagens de forma conjunta. Da mesma forma, o capítulo sobre *Processos e Práticas da Engenharia de Requisitos* também pode ser utilizado como revisão bibliográfica, onde a maior contribuição é a uniformização de conceitos sobre engenharia de requisitos utilizados na documentação das abordagens CMMI e RUP.

6.3 Possibilidade de Trabalhos Futuros

Embora este trabalho tenha atingido seu marco final, com a institucionalização do processo proposto, as atividades relacionadas à avaliação de resultados e melhoria de processos na organização continuam sendo executadas. O processo continuará evoluindo de forma controlada, adaptando-se às características de projetos desenvolvidos na organização e tendo seu repositório de orientações e melhores práticas em constante crescimento.

Uma oportunidade de desenvolvimento complementar é integrar, às descrições do processo de engenharia de requisitos, um plano detalhado de métricas relacionadas a requisitos. Após a definição de tal plano, a coleta

destas métricas deve ser feita em todos os projetos da organização, de forma a criar bases históricas que servirão de apoio a decisões sobre o processo no futuro.

Outro ponto que pode ser trabalhado em estudos futuros é a revisão detalhada das metas e práticas genéricas definidas no CMMI para os níveis de maturidade 2 e 3. Estas metas e práticas, embora consideradas durante a definição do processo de engenharia de requisitos proposto, não foram associadas a cada uma das atividades do processo, como foi o caso com as metas e práticas específicas.

REFERÊNCIAS

ANDREA, J. et al. **Agile Alliance Organization**. 2005. Disponível em: <http://www.agilealliance.org/>. Acesso em: dez. 2005.

AMBLER, S. W. **Agile Modeling and the Unified Process**. 2001. Disponível em: <http://www.agilemodeling.com/essays/agileModelingRUP.htm>. Acesso em: dez. 2005.

AMBLER, S. W. **Agile Modeling:** Effective Practices for Extreme Programming and the Unified Process. New York: John Wiley & Sons, 2002.

AMBLER, S. W. **Agile Requirements Best Practices**. Oct. 30, 2005. Disponível em: <http://www.agilemodeling.com/essays/agileRequirementsBest

Practices.htm>. Acesso em: dez. 2005.

ARMOUR, F.; MILLER, G. **Advanced Use Case Modeling**. [S.l.]: Addison-Wesley, 2001.

AVISON, D. E.; FITZGERALD, G. Where Now for Development Methodologies? **Communications of the ACM**, New York, v.46. n.1, p. 79-82, Jan. 2003.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H.D. The Goal Question Metric Approach. In: MARCINIAK, J. J. (Ed.). **Encyclopedia of Software Engineering**. [S.l.]: John Wiley & Sons, 1994. v.2, p. 528-532. Disponível em: <http://www.cs.umd.edu/users/mvz/handouts/gqm.pdf>. Acesso em: dez. 2005.

BECK, K. et al. **Manifesto for Agile Software Development**. Agile Alliance. 2001. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: dez. 2005.

BECK, K. **Extreme Programming Explained:** Embrace Change. [S.l.]: Addison-Wesley Longman, 2000.

BITTNER, K.; SPENCE I. **Use Case Modeling**. [S.l.]: Addison-Wesley, 2002.

BOEHM, B. A Spiral Model of Software Development and Enhancement. In: INTERNATIONAL WORKSHOP ON SOFTWARE PROCESS AND SOFTWARE ENVIRONMENTS, 1985, Trabuco Canyon, US. **Proceedings…** [S.l. : s.n.], 1985.

BOEHM, B. Software Risk Management: Principles and Practices. **IEEE Software**, Los Alamitos, v.8, n.1, p.32-41, Jan. 1991.

BOEHM, B.; TURNER, R. Using Risk to Balance Agile and Plan-Driven Methods. **Computer**, New York, v.36, n.6, p. 57-66, June 2003.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Reading, MA: Addison Wesley, 1999.

CLEGG, C. et al. **The Performance of Information Technology and the Role of Human and Organizational Factors**. UK: Economic and Social Reseach Council, 1996.

COCKBURN, A. **Crystal Clear:** A Human-Powered Methodology for Small Teams. [S.l.]: Addison-Wesley, 2004.

DAL'OSTO, F. **Utilizando CMM e GQM na Avaliação e Classificação de Ambientes de Desenvolvimento de Software**. 2001. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

DAMIAN, D. et al. An industrial case study of immediate benefits of requirements engineering process improvement at the Australian Center for Unisys Software. **Empirical Software Engineering**, [S.l.], p. 45-75, Mar. 2004.

DORFMAN, M.; THAYER, R. H. **Standard, guidelines and examples on system and software requirements engineering**. Los Alamitos: IEEE Computer Society Press, 1990.

FAZACKERLEY, B. et al. **About the DSDM Consortium and the Framework**. Disponível em: <http://www.dsdm.org/>. Acesso em: dez. 2005.

DUTTON, J.; SVERDRUP, J. A CMMI Case Study: Process Engineering vs. Culture and Leadership. **The Journal of Defense Software Engineering**, [S.l.], Dec. 2002. Disponível em: <http://software-engineer.org/downloads/

dutton.pdf>. Acesso em: dez. 2005.

EVANS, G. A simplified approach to RUP. **IBM developerWorks**, [S.l.], 18 Nov. 2003. Disponível em: <http://www-128.ibm.com/developerworks/rational/library/

354.html>. Acesso em: dez. 2005.

FITZGERALD, B.; O'KANE, T. A Longitudinal Study of Software Process Improvement. **IEEE Software**, New York, v.16, n.3, p.37-45, May-June 1999.

FITZGERALD, B.; RUSSO, N. L.; O'KANE, T. Software Development Method Tailoring at Motorola. **Communications of the ACM**, New York, v.46, n.4, p. 65-70, Apr. 2003.

FOWLER, M. **Is Design Dead?**. May 2004. Disponível em: <http://www.martinfowler.com/articles/designDead.html>. Acesso em: dez. 2005.

GALLAGHER, B.; BROWNSWORD, L. **The Rational Unified Process and the Capability Maturity Model – Integrated Systems/Software Engineering**. Pittsburgh, MA: ESEPG, Software Engineering Institute, Carnegie Mellon University, 2001. RUP/CMMI Tutorial.

GRADY, R. **Practical Software Metrics for Project Management & Process Improvement**. Englewood Cliffs: Prentice-Hall, 1992.

HARTER, D.; KRISHNAN, M.; SLAUGHTER, S. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. **Management Science**, Evanston, IL, v.46, n.4, p. 451-466, Apr. 2000.

HARTMANN, J.; PRICE, R. T. **Utilizando padrões organizacionais e avaliação de risco para adaptar a metodologia de desenvolvimento de *software***. 2004. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

HEBERT, M.; BENBASAT, I. Adopting Information Technology in Hospitals: The Relationships between Attitudes/Expectations and Behavior. **Hospital and Health Services Administration**, [S.l.], v.39, n.3, p.369-383, 1994.

HIGHSMITH, J. A. III. **Adaptive Software Development:** A Collaborative Approach to Managing Complex Systems. New York: Dorset House Publishing, 1999.

IBM RATIONAL CORPORATION. **Rational Unified Process**: 2003.06.13. [S.l.], 2003.

IBM RATIONAL CORPORATION. **Rational ClearCase**. [S.l.], 2005. Disponível em: <http://www-306.ibm.com/software/awdtools/clearcase/>. Acesso em: dez. 2005.

IBM RATIONAL CORPORATION. **Rational RequisitePro**. [S.l.], 2005. Disponível em: <http://www-306.ibm.com/software/awdtools/reqpro/>. Acesso em: dez. 2005.

IEEE. **IEEE Standard 830:** guide to software requirements specifications. New York, 1984.

IEEE. **IEEE Standard 830:** recommended practice for software requirements specifications. New York, 1998.

IEEE. **IEEE Standard Computer Dictionary:** A Compilation of IEEE Standard Computer Glossaries. New York, NY, 1990.

JACOBSON, I. et al. **Object-Oriented Software Engineering:** A Use Case Driven Approach. Reading, MA: Addison-Wesley, 1992.

JEFFRIES, R. **Extreme Programming and the Capability Maturity Model**. 2000. Disponível em: <http://www.xprogramming.com/xpmag/xp_and_cmm.

htm>. Acesso em: dez. 2005.

JEFFRIES, R. **Essential XP:** Card, Conversation, Confirmation. 2001. Disponível em: <http://www.xprogramming.com/xpmag/expCardConversation

Confirmation.htm>. Acesso em: dez. 2005.

JIROTKA, M.; GOGUEN, J. **Requirements Engineering – Social and Technical Issues**. San Diego, CA: Academic Press, 1994.

KRUCHTEN, P. **The Rational Unified Process:** An Introduction. 2nd ed. Reading, MA: Addison-Wesley, 2001.

KRUCHTEN, P. Agility with the RUP. **Cutter IT Journal, The Journal of Information Technology Management**, [S.l.], v.14, n.12, p. 27 – 33, Dec. 2001.

LARMAN, C.; BASILI, V.R. Iterative and Incremental Development: A Brief History. **Computer**, New York, v.36, n.6, pp.47-55, June 2003.

LEFFINGWELL, D.; WIDRIG, D. **Managing Software Requirements, A Unified Approach**. Upper Saddle River, NJ: Addison-Wesley, 2000.

LYCETT, M. et al. Migrating Agile Methods to Standardized Development Practice. **Computer**, New York, v.36, n.6, p. 79-85, June 2003.

MACAYLAY, L. A. **Requirements Engineering**. London, UK: Springer–Verlag, 1997.

MANZONI, L.; PRICE, T. Analyzing RUP (Rational Unified Process) Regarding the Satisfaction of CMM (Capability Maturity Model) Levels 2 and 3. In: ARGENTINE SYMPOSIUM ON SOFTWARE ENGINEERING, SADIO, 2001, Buenos Aires, Argentina. **Proceedings…** Buenos Aires: SADIO, 2001. p.135-144.

MANZONI, L.; PRICE, T. **Uso de Sistema de Gerência de Workflow para Apoiar o Desenvolvimento de Software Baseado no Processo Unificado da Rational Estendido para Alcançar Níveis 2 e 3 do Modelo de Maturidade**. 2001. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MANZONI, L.; PRICE, T. Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Levels 2 and 3. **IEEE Transactions on Software Engineering**, Los Alamitos, v.29, n.2, p.181-192, 2003.

MARTIN, R. C.; BOOCH, G.; NEWKIRK, M. J. **The Process.** 1998. Disponível em: <www.objectmentor.com/publications/RUPvsXP.pdf>. Acesso em: dez. 2005. Capítulo preliminar do livro Object Oriented Analysis and Design with Applications, 2. ed., Addision Wesley Longman, 1998.

MCCOY, D.; MARTIN, K. **Process Improvement in an Organization on Steroids:** How to Maintain Quality as Your Organization Grows and Spreads Rapidly. Seattle, Washington: [s.n.], 2005.

PALMER, S.; FELSING, J. **Practical Guide to Feature-Driven Development**. [S. l.]: Prentice Hall, 2002.

PAULK, M. C. et al. **Key Practices of the Capability Maturity Model, Version 1.1**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993. (Technical Report: CMU/SEI-93-TR-025).

PAULK, M. C. Extreme Programming from a CMM Perspective. **IEEE Software**, Los Alamitos, v.18, n.6, p.19-26, Nov./Dec. 2001. Disponível em: <http://www.agilealliance.org/articles/articles/XPFromACMMPerspective-Paulk.pdf>. Acesso em: dez. 2005.

POHL, K. The three dimensions of Requirements Engineering. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAiSE, 5., 1993 **Proceedings…** Paris: Springer-Verlag, 1993. p.175-292.

RATIONAL UNIVERSITY. **Fundamentals of Rational RequisitePro, Student Manual, Version 2002.05.00**. Cupertino, CA, 2001. Revised Mar. 2002.

RATIONAL UNIVERSITY. **Requirements Management with Use Cases, Student Manual, Version 2002.05.00**. Cupertino, CA, 2001. Revised Mar. 2002.

REIFER, D. J. XP and the CMM. **IEEE Software**, Los Alamitos, v.20, n.3, p.14-15, May/June 2003.

REITZIG, R. W. **Using Rational Software Solutions to Achieve CMMI Level 2**. 2003. Disponível em: <http://www.cognence.com/CMMI_TheRationalEdge

_Jan2003.pdf>. Acesso em: dez. 2005.

RIEMENSCHNEIDER, C. K.; HARDGRAVE, B. C; DAVIS, F. D. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five

Theoretical Models. **IEEE Transactions on Software Engineering**, Los Alamitos, v.28, n.12, p.1135-1145, December 2002.

ROSEMBERG, D. **Use Case Driven Object Modeling with UML, A Practical Approach**. Upper Saddle River, NJ: Addison-Wesley, 1999.

ROYCE, W. Managing the Development of Large Software System. In: IEEE WESCON, 1970. **Proceedings…** [S.l.:s.n.], 1970. p.1-9.

ROYCE, W. **CMM vs. CMMI**: From Conventional to Modern Software Management. 2002. Disponível em: <http://www.therationaledge.com/content/

feb_02/f_conventionalToModern_wr.html>. Acesso em: dez. 2005.

SANTOS, J. B. **Extraindo o melhor de XP, Agile Modeling e RUP para melhor produzir software**. Rio de Janeiro: Vice-Reitoria Administrativa, PUC-Rio, 2002.

SCHWABER, K. **The Impact of Agile Processes on Requirements Engineering**. 2002. Disponível em: <http://www.agilealliance.org/articles/

schwaberkentheimpacto/file>. Acesso em: dez. 2005.

SCHWABER, K. **Agile Project Management with Scrum**. Redmond, Washington, USA: Microsoft Press, 2004.

SOFTWARE ENGINEERING INSTITUTE. **Capability Maturity Model Integration (CMMI), Version 1.1, CMMI for Software Engineering (CMMI-SW, V1.1), Staged Representation**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. Disponível em: <http://www.sei.cmu.edu/cmmi/models/sw-staged.doc>. Acesso em: dez. 2005.

SOFTWARE ENGINEERING INSTITUTE. **Introduction to Capability Maturity Model Integration (CMMI) Version 1.1, Staged Representation**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

SOFTWARE ENGINEERING INSTITUTE. **Process Maturity Profile, CMMI v1.1, SCAMPI v1.1 Class A Appraisal Results**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. (CMMI Appraisal Program, 2005 Mid-Year Update).

SOFTWARE ENGINEERING INSTITUTE. **Frequently Asked Questions (FAQs)**. 2005. Disponível em: <http://www.sei.cmu.edu/cmmi/faq/term-faq.html>. Acesso em: dez. 2005.

SMITH, J. **The estimation of Effort Based on Use Cases.** 1999. Disponível em: <http://www.bfpug.com.br/Artigos/UCP/Smith-The_Estimation_of_Effort_ Based_on_Use_Cases.pdf>. Acesso em: dez. 2005.

SMITH, J. **Reaching CMM Levels 2 and 3 with the Rational Unified Process**. 2000. Disponível em: <http://www3.software.ibm.com/ibmdl/pub/software/ rational/web/whitepapers/2003/rupcmm.pdf>. Acesso em: dez. 2005.

SOMMERVILLE, I. **Software Engineering**. 4th ed. USA: Addison-Wesley, 1992.

SOMMERVILLE, I. **Requirements Engineering, An Overview**. UK: Lancaster University, 2001. (Class Presentation).

TAFT, D. K. Microsoft Taps Former Rational Heavyweight to Lend Credence to Enterprise Tools Play. **eWeek.coom,** [S.l.], Nov. 2005. Disponível em: <http://www.eweek.com/article2/0,1895,1886531,00.asp>. Acesso em: dez. 2005.

TYSON, B.; BROWNSWORD, L.; BROWNSWORD, R. **Leveraging RUP and CMMI for Real-World Successes**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. Disponível em: <http://www.numbersix.com/csdi/documents/ESEPG-CMMIandRUP-Final.pdf>. Acesso em: dez. 2005.

WIEGERS, K. E. When Telepathy Won't Do: Requirements Engineering Key Practices. **Cutter IT Journal**, [S.l.], May 2000. Disponível em: <http://www.processimpact.com/articles/telepathy.html>. Acesso em: dez. 2005.

WIEGERS, K. E. **Software Process Improvement Handbook:** A Practical Guide. 2005. Disponível em: <http://www.processimpact.com/pubs.shtml>. Acesso em: dez. 2005.

WIERINGA, R. **An Introduction to Requirements Traceability**. [S.l. : s.n.], 1995.

WILLIAMS, L.; COCKBURN, A. Agile Software Development: It's about Feedback and Change. **Computer**, New York, v.36, n.6, p. 39-43, June 2003.

ZAVE, P. Classification of Research Efforts in Requirements Engineering. **ACM Computing Surveys**, New York, v.29, n.4, p.315-321, Dec. 1997.

ZUBROW, D. **Measuring Software Product Quality:** the ISSO 25000 Series and CMMI. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. Disponível em: <http://www.sei.cmu.edu/sema/presentations/zubrow/esepg/esepg.pdf>. Acesso em: dez. 2005.