PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO SANTOS DE ESPINDOLA

# UMA ARQUITETURA DE INFORMAÇÃO PARA GERÊNCIA DE REQUISITOS EM DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

PORTO ALEGRE
2006

RODRIGO SANTOS DE ESPINDOLA

# UMA ARQUITETURA DE INFORMAÇÃO PARA GERÊNCIA DE REQUISITOS EM DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

Dissertação apresentada como requisito parcial a obtenção do grau de Mestre em Ciência da Computação, pelo Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: PROF. DR. JORGE LUIS NICOLAS AUDY

PORTO ALEGRE
2006

# Dados Internacionais de Catalogação na Publicação (CIP)

E77a Espindola, Rodrigo Santos de
Uma arquitetura de informação para gerência de requisitos em desenvolvimento distribuído de software / Rodrigo Santos de Espindola. – Porto Alegre, 2006.
125 fls.

Diss. (Mestrado em Ciência da Computação) – Fac. de Informática, PUCRS.
Orientação: Prof. Dr. Jorge Luis Nicolas Audy

1. Informática. 2. Engenharia de Software 3. Sistemas Distribuídos. I. Audy, Jorge Luis Nicolas.

CDD 005.1

Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS

Av. Ipiranga, 6681 - Prédio 16
CEP 90619-900 - Porto Alegre - RS - Brasil
http://www.pucrs.br/biblioteca

Fone: 0 (xx) 51 3320-3544
FAX: 0 (xx) 51 3320-3548
e-mail: bceadm@pucrs.br

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Reconhecido pelo Parecer No. 930/98.C.N.E. Homologação Publicada no D.O.U. de 30/12/98.

## TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Uma Arquitetura de Informação para Gerência de Requisitos em Desenvolvimento Distribuído de Software***", apresentada por Rodrigo Santos de Espindola, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 30/03/2006 pela Comissão Examinadora:

Prof. Dr. Jorge Luis Nicolas Audy – PPGCC/PUCRS
Orientador

Prof. Dr. Marcelo Blois Ribeiro – PPGCC/PUCRS

Prof. Dr. Ricardo Melo Bastos – PPGCC/PUCRS

Prof. Dr. Marcelo Soares Pimenta – UFRGS

Homologada em 02/05/06, conforme Ata No. 19 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

AV. IPIRANGA, 6681 - PRÉDIO 16/SALA 106
90619-900 - PORTO ALEGRE -RS -BRASIL
TELEFONE: 55 51 33203611 FAX 55 51 3320.3621
http: //www.pucrs.br/inf/pos e-mail: ppgcc@inf.pucrs.br

# AGRADECIMENTOS

Agradeço ao Prof. Dr. Jorge Audy, pela dedicação, paciência e objetividade com que me orientou ao longo destes anos. Aprendi mais sobre a pesquisa científica em nossas reuniões do que em todas as aulas que já assisti sobre o tema.

Agradeço ao Prof. Dr. Toacy Cavalcanti, por todo apoio durante o estágio de docência.

Agradeço aos professores Ricardo Bastos e Marcelo Blois, pelas dicas e pela criteriosa avaliação durante todas as etapas do desenvolvimento desta pesquisa.

Agradeço a Luciana Spagnoli, por sua eterna disposição em ajudar os amigos. Seu vasto conhecimento sobre as normas do PPGCC e sua valiosa (e sempre gratuita) consultoria, poupou-me muito tempo e dor de cabeça.

Agradeço ao colega Leandro Lopes, pelo apoio, coleguismo e intermináveis diálogos sobre os mais variados temas. Agradeço também por todo o material de pesquisa (e literário) que me forneceu.

Agradeço ao colega Azriel Majdenbaum, pela amizade e pelo apoio à pesquisa realizada nestes dois anos. Sem seu empenho em abrir portas esta pesquisa teria demandado um esforço muito maior.

Por fim, o agradecimento mais importante. Agradeço a minha amada esposa Anete. Ela é a pessoa que dá sentido a este e a todos os esforços que faço em minha vida. Ela é quem faz qualquer desafio valer a pena.

*"A mente que se abre para uma nova idéia jamais voltará a seu tamanho original".*
*(Albert Einstein)*

# RESUMO

A distribuição das equipes de desenvolvimento tem provocado diversos desafios ao processo de software. Dentre os desafios, a engenharia de requisitos (ER) destaca-se, sofrendo impacto de fatores como distância, diferenças culturais e de fuso-horário, bem como limitações dos meios de comunicação disponíveis. Nesse contexto, o gerenciamento das informações relacionadas a requisitos torna-se crítico para garantir que as informações necessárias sobre um determinado domínio ou aplicação estão disponíveis para as equipes de desenvolvimento geograficamente dispersas e que estas informações sejam organizadas de forma a permitir futuro acesso por projetos de manutenção no mesmo escopo. Nesse sentido, esta dissertação de mestrado tem como objetivo propor um modelo de arquitetura de informação para gerência de requisitos em desenvolvimento distribuído de software (DDS). Esta proposta visa contribuir para a área de engenharia de software ao preencher uma lacuna existente na área de DDS, especificamente no que se refere à ER e à manutenção de software. Além disso, este estudo apresenta novos dados empíricos e busca contribuir também para a pesquisa na área de engenharia de software através do emprego de métodos qualitativos de pesquisa científica. O método de pesquisa utilizado foi o estudo de caso e a base empírica da pesquisa envolve uma unidade de desenvolvimento de software de uma empresa multinacional de grande porte localizada no Brasil.

Palavras-chave: Engenharia de Software. Processo de Software. Desenvolvimento Distribuído de Software. Desenvolvimento Global de Software. Engenharia de Requisitos. Manutenção de Software.

# ABSTRACT

The distributed software development (DSD) has caused several challenges to the software development process. Among these challenges, the requirement engineering (RE) is highlighted due to the impact of factors like distance, cultural and time zone differences, and communications restrictions. In this context, the management of requirements related information is turned critical to assure that the required information about a specific domain or application will be available to the software development teams geographically distributed and it will also be available to the future maintenance projects. This way, the goal of this master thesis is to propose a structural model to requirements management in DSD. This proposal aims to contribute to the software engineering filling an existing gap in the DSD area, specifically about the RE and the software maintenance. Besides, this study shows new empirical data and also aims to contribute to software engineering through applying qualitative research methods. The research method adopted was case study, conducted in a software development unit of a multinational organization located in Brazil.

Keywords: Software Engineering, Software Process, Distributed Software Development, Global Software Development, Requirements Engineering, Software Maintenance.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE SIGLAS

| | |
|---|---|
| **CD** | *Compact Disk* |
| **DDS** | Desenvolvimento Distribuído de Software |
| **DGS** | Desenvolvimento Global de Software |
| **DM** | *Delivery Manager* |
| **ER** | Engenharia de Requisitos |
| **EUA** | Estados Unidos da América |
| **FCS** | Fatores Críticos de Sucesso |
| **FP** | *Function Points* |
| **IEEE** | *Institute of Electrical and Electronic Engineers* |
| **ISO** | *International Organization for Standardization* |
| **LOC** | *Lines of Code* |
| **MSF** | *Microsoft Solutions Framework* |
| **P2P** | Ponto-a-Ponto |
| **PM** | *Project Manager* |
| **PMI** | *Project Management Institute* |
| **PPGCC** | Programa de Pós-Graduação em Ciência da Computação |
| **PUCRS** | Pontifícia Universidade Católica do Rio Grande do Sul |
| **RUP** | *Rational Unified Process* |
| **SA** | *System Architect* |
| **SEPG** | *Software Engineering Process Group* |
| **SGBD** | Sistemas de Gerenciamento de Banco de Dados |
| **SPICE** | *Software Process Improvement and Capability dEtermination* |
| **SRS** | *Software Requirements Specification* |
| **SW-CMM** | *Software Capability Maturity Model* |
| **TL** | *Tech Lead* |
| **TSL** | *Test Lead* |

# SUMÁRIO

# 1 Introdução

Atualmente, é cada vez mais significativo o número de empresas que estão distribuindo seus esforços de desenvolvimento e manutenção de software ao redor do mundo. Desta forma, tem-se criado um cenário onde projetos de software são desenvolvidos com equipes distribuídas, caracterizando assim o desenvolvimento distribuído de software (DDS). Segundo Carmel [CAR99], as principais características que diferenciam DDS do desenvolvimento co-localizado são: dispersão geográfica, dispersão temporal e diferenças culturais. Estas características têm criado uma nova classe de problemas a serem resolvidos pelos pesquisadores da área de engenharia de software. Na indústria, os engenheiros de software também têm reconhecido a grande influência desta nova forma de trabalho e estão em busca de modelos que facilitem o desenvolvimento de software com equipes distribuídas. Este crescente interesse, tanto na área acadêmica quanto na indústria, tem motivado diversas pesquisas sobre o tema, tais como as apresentadas por [CAR99], [EVA00], [PRI04] e [EVA05].

Do ponto de vista de processos de engenharia de software, a engenharia de requisitos (ER) é tida como um dos principais desafios para projetos desenvolvidos neste contexto. A ER demanda atividades fortemente baseadas na comunicação e na transferência de conhecimento, sendo, portanto, diretamente afetada pela dispersão geográfica, dispersão temporal e pelas diferenças culturais entre diversas localidades envolvidas em projetos distribuídos. Assim, em projetos distribuídos percebe-se uma grande dificuldade das organizações em elicitar, analisar, documentar e gerenciar corretamente os requisitos. Desta forma, alguns estudos foram desenvolvidos nos últimos anos ([DAM02], [ZOW02], [PRI03], [LOP03], [LOP03b], [LOP04], [ESP05], [ESP05b] e [SAY05]) buscando entender quais são as principais dificuldades presentes na ER, no cenário específico de DDS.

Já no contexto da manutenção de software, a ER torna-se especialmente importante. Muitos dos sistemas de software desenvolvidos nas últimas décadas continuam sendo utilizados até os dias atuais. Estes sistemas, geralmente chamados de Sistemas Legados, são definidos como sistemas antigos, freqüentemente mal projetados e documentados, críticos para a organização e que devem ser suportados por muitos anos [PRE01]. Entretanto, características típicas dos sistemas legados, tais como ausência dos desenvolvedores originais e ausência de uma documentação consistente, tornam difícil, em projetos de manutenção de software, o trabalho de ER. Dentre os processos de ER, o processo de gerência de requisitos é significativamente afetado por tais características, tendo em vista que sua finalidade é justamente a manutenção das especificações de requisitos. Por conseqüência, várias atividades relacionadas à análise do

problema, análise de impacto, atualização da documentação e os testes das modificações realizadas também são dificultados. Estas dificuldades têm impacto direto no sucesso de tais projetos, bem como na qualidade dos produtos resultantes do processo de manutenção.

Portanto, se considerarmos que, segundo [ZOW02] e [DAM02], os problemas fundamentais da ER são exacerbados quando as equipes de desenvolvimento de software estão distribuídas geograficamente, então os assuntos gerência de requisitos, DDS e manutenção de software tornam-se temas de relevância para a pesquisa acadêmica e aplicada. Com base nesta contextualização, o tema escolhido para dissertação de mestrado é a gerência de requisitos em ambientes de DDS.

## 1.1 Questão de Pesquisa

A questão de pesquisa abordada no projeto de pesquisa aqui apresentado foi a seguinte:

*Como desenvolver a gerência de requisitos em projetos de manutenção de software conduzidos em ambientes de desenvolvimento distribuído de software?*

## 1.2 Objetivos

Com base na contextualização apresentada na seção anterior, o objetivo geral desta pesquisa é propor um modelo de arquitetura de informação para gerência de requisitos em DDS.

Para atender a este objetivo geral emergem os seguintes objetivos específicos:

- Aprofundar o conhecimento em gerência de requisitos, manutenção de software, DDS e tópicos relacionados;
- Identificar o processo de ER utilizado por uma organização em projetos de manutenção de software em ambientes DDS e as principais dificuldades enfrentadas neste contexto;
- Elaborar uma proposta preliminar de modelo de arquitetura de informação para gerência de requisitos em DDS;
- Verificar a aplicabilidade do modelo proposto e os benefícios decorrentes de sua utilização.

## 1.3 Organização do volume

Este volume está organizado em oito capítulos. No capítulo 2 é apresentado o referencial teórico desta pesquisa, envolvendo os principais conceitos e áreas do estudo: processo de software, manutenção de software, ER e DDS. A apresentação deste referencial teórico é feita de

forma abrangente, em virtude da natureza exploratória desta pesquisa e do emprego de métodos qualitativos, que determinam a necessidade de uma larga e consistente fundamentação teórica [YIN01].

No capítulo 3 são apresentados os trabalhos relacionados. Foram selecionados artigos sobre estudos em áreas relacionadas ou complementares a pesquisa aqui apresentada. No capítulo 4, apresenta-se o método de pesquisa, descrevendo cada uma das etapas do estudo e apresentando o desenho de pesquisa em detalhe.

No capítulo 5 é apresentado o estudo de caso 1. Este estudo de caso teve como finalidade identificar o processo de ER seguido pela organização em projetos de manutenção de software e os principais problemas enfrentados neste contexto. Após a execução deste estudo de caso, foi elaborada, na segunda fase desta pesquisa, a proposta preliminar de modelo de arquitetura de informação para gerência de requisitos em DDS. Esta proposta foi elaborada tendo em vista as dificuldades identificadas através da pesquisa bibliográfica e do estudo de caso 1.

No capítulo 6 é apresentado o modelo de arquitetura de informação para gerência de requisitos em DDS proposto nesta pesquisa. Este modelo de arquitetura foi elaborado tendo em vista as dificuldades identificadas através da pesquisa bibliográfica e do estudo de caso 1, executados na primeira fase desta pesquisa, e foi refinado com base nos resultados do estudo de caso 2.

No capítulo 7 é apresentado o estudo de caso 2. Este estudo de caso teve como finalidade avaliar a aplicabilidade do modelo de arquitetura de informação para gerência de requisitos em DDS e os benefícios decorrentes de sua utilização em projetos reais conduzidos em um ambiente DDS. Este estudo de caso também identificou oportunidades de melhoria na proposta preliminar.

Por fim, no capítulo 8, são apresentadas as conclusões deste volume. São descritas as colaborações deste estudo, bem como suas limitações e trabalhos futuros.

# 2 Base Teórica

O referencial teórico representa uma importante etapa em estudos de base qualitativa [YIN01], tais como a pesquisa apresentada neste volume. Neste capítulo será apresentada a base teórica utilizada nesta pesquisa.

A seção 2.1 apresenta a base teórica sobre processo de software, incluindo definição, fases, a norma da *International Organization for Standardization* (ISO) sobre processo de software e dois dos principais processos de software utilizados pela indústria na atualidade. A seção 2.2 apresenta a base teórica sobre manutenção de software, incluindo os principais conceitos envolvidos, a norma ISO sobre manutenção de software, uma breve análise sobre a relação entre a manutenção de software e os processos de software descritos na seção 2.1 e alguns aspectos gerencias da manutenção de software. A seguir, a seção 2.3 apresenta a base teórica sobre ER, incluindo os principais conceitos e processos envolvidos no tema. Por fim, a seção 2.4 apresenta o referencial teórico sobre DDS, incluindo a relação a relação entre DDS, manutenção de software e ER.

## 2.1 Processo de Software

O principal objetivo da engenharia de software é melhorar a qualidade do software desenvolvido. A qualidade do produto de software, entretanto, está fortemente relacionada à qualidade do processo de software [FUG00]. Para muitos engenheiros de software a qualidade do processo de software é tão importante quanto a qualidade do produto. Abordagens importantes como as normas ISO 9000, ISO/IEC 12207, *Capability Maturity Model* (CMM) e o *Software Process Improvement and Capability dEtermination* (SPICE) sugerem que melhorando o processo de software, podemos melhorar a qualidade dos produtos de software [ROC01].

Nesta seção serão apresentados os principais conceitos relacionados ao processo de software. Como referencial teórico para os processos do ciclo de vida de software será apresentada a norma ISO/IEC 12207. A seguir, o *Rational Unified Process* (RUP) será apresentado devido a sua relevância como processo de engenharia de software e sua ampla utilização no mercado. Ao final será apresentado o *Microsoft Solutions Framework* (MSF), por ser o processo utilizado na organização onde a pesquisa deste autor foi realizada durante o curso de mestrado.

### 2.1.1 Definição

O processo de software [HUM89] pode então ser definido como o conjunto de atividades,

métodos e práticas que guiam a produção de software. Um processo efetivo deve considerar a relação das tarefas necessárias, as ferramentas e métodos, e as habilidades, treinamento e motivação das pessoas envolvidas.

Em [PRE01] o processo de software é definido como um *framework* para as tarefas que são necessárias para a construção de sistemas de alta qualidade e é visto, do ponto de vista da engenharia de software, como um elo entre aspectos tecnológicos. As áreas chave de processo formam a base para o controle gerencial dos projetos de software e estabelece um contexto no quais métodos e técnicas são aplicadas, artefatos (modelos, documentos, dados, relatórios, formulários e etc.) são produzidos, *milestones* são estabelecidos, a qualidade é garantida e as alterações são apropriadamente gerenciadas.

### 2.1.2 Fases

Independentemente do paradigma escolhido, da área de aplicação, do tamanho do projeto ou de sua complexidade o processo de desenvolvimento pode ser caracterizado em três fases genéricas [ROC01]: a definição, o desenvolvimento e a manutenção.

A fase de definição tem como objetivo definir 'o quê', ou seja, quais informações serão processadas, quais funções e desempenho são desejados, quais interfaces devem ser estabelecidas, quais restrições do projeto e critérios de validação são necessários. Nesta fase, três etapas específicas são sempre realizadas: a análise do sistema, o planejamento do projeto e a análise dos requisitos.

A fase de desenvolvimento tem como objetivo o 'como', ou seja, como devem ser projetadas as estruturas de dados e a arquitetura do software, como os procedimentos devem ser implementados, como o projeto deve ser traduzido para uma linguagem de programação, como o teste deve ser executado, etc. Nesta fase também são sempre realizadas três etapas específicas: o projeto, a codificação e o teste do software.

A fase de manutenção tem como objetivo as mudanças, sejam elas decorrentes de erros (manutenção corretiva), adaptações necessárias decorrentes de alterações no meio ambiente (manutenção adaptativa) e melhoramentos relacionados às novas necessidades do usuário (manutenção perfectiva).

### 2.1.3 A norma ISO/IEC 12207

A globalização da economia tem influenciado as empresas produtoras e prestadoras de serviços de software a alcançar o patamar de qualidade e produtividade internacional para enfrentarem a competitividade cada vez maior. De acordo com [ROC01], a norma internacional ISO/IEC 12207 - *Information Technology – Software Life Cycle Processes* é usada como referência em

muitos países, inclusive Brasil, para alcançar esse diferencial competitivo. Ela tem por objetivo auxiliar os envolvidos na produção de software a definir seus papéis, por meio de processos bem definidos, e assim proporcionar às organizações que a utilizam um melhor entendimento das atividades a serem executadas nas operações que envolvem, de alguma forma, o software.

Segundo [ROC01], a norma é flexível para as abordagens de engenharia de software envolvidas, sendo utilizável em qualquer modelo de ciclo de vida (em cascata, incremental, evolutivo, etc.), com qualquer método ou técnica de engenharia de software (projeto orientado a objetos, técnicas estruturadas, prototipação, etc.) e com quaisquer linguagens de programação (*Cobol, Ada, Visual Basic*, etc.). Estas escolhas dependerão do projeto e do estado da arte da tecnologia e serão deixadas a critério dos usuários da norma.

#### *2.1.3.1 Arquitetura da Norma*

Para [ROC01] a arquitetura descrita na norma utiliza uma terminologia bem definida composta de processos, atividades e tarefas para a aquisição, fornecimento, desenvolvimento, operação e manutenção de software. A norma estabelece uma arquitetura de alto nível para o ciclo de vida de software que abrange desde a concepção até a descontinuidade do mesmo. Esta arquitetura é baseada em processos chave e no inter-relacionamento entre eles e segue dois princípios básicos [ISO98]:

- **Modularidade:** os processos têm alta coesão e baixo acoplamento, ou seja, todas as partes de um processo são fortemente relacionadas e o número de interfaces entre os processos é mantido ao mínimo.
- **Responsabilidade:** cada processo na norma é de responsabilidade de uma 'parte envolvida', que pode ser uma organização ou parte dela. As partes envolvidas podem ser de uma mesma organização ou de organizações diferentes. A norma considera os termos 'parte' e 'organização' como sinônimos. Além disto, o tamanho da organização pode variar de uma a várias pessoas.

Na norma, os processos que envolvem o ciclo de vida do software são agrupados em três classes que representam sua natureza. Cada processo é definido em termos de suas próprias atividades, e cada atividade é adicionalmente definida em termos de suas tarefas [ISO98].

#### *2.1.3.2 Processos*

Segundo [ROC01], os **Processos Fundamentais** atendem ao início, à contratação entre o adquirente e o fornecedor, à execução do desenvolvimento, da operação e da manutenção de produtos de software durante o ciclo de vida do software. São eles:

- **Processo de aquisição:** Define as atividades do adquirente (organização que adquire um sistema ou produto de software). Inicia-se com a definição da necessidade de adquirir um sistema, um produto ou serviço de software e continua com a preparação e a emissão de pedido de proposta, com a seleção de fornecedores e da gerência do processo de

aquisição mediante a aceitação do sistema, produto ou serviço de software.

- **Processo de fornecimento:** Define as atividades do fornecedor (organização que fornece o produto de software ao adquirente). O processo pode ser iniciado tanto pela decisão de preparar uma proposta para atender à solicitação de um adquirente, quanto pela assinatura e celebração de um contrato com o adquirente para fornecer o sistema, produto ou serviço de software. O processo continua com a determinação dos procedimentos e recursos necessários para gerenciar e garantir o projeto, incluindo o desenvolvimento e a execução dos planos de projeto até a entrega do sistema, produto ou serviço de software para o adquirente.
- **Processo de desenvolvimento:** Define as atividades do desenvolvedor (organização que define e desenvolve o produto de software). Inclui atividades para a análise de requisitos, projeto, codificação, integração, teste, instalação e aceitação relacionadas ao produto de software.
- **Processo de operação:** Define as atividades do operador (organização que provê o serviço de operação de um sistema computacional no seu ambiente de funcionamento para seus usuários). As atividades deste processo envolvem a operação do produto de software e o suporte operacional aos usuários.
- **Processo de manutenção:** Define as atividades do mantenedor (organização que provê os serviços de manutenção do software, isto é, o gerenciamento das modificações no software para mantê-lo atualizado e em perfeita operação). Esse processo é ativado quando o produto de software é submetido a modificações no código e na documentação associada, devido a um problema ou à necessidade de melhoria ou adaptação. O objetivo é modificar um produto de software existente, preservando a sua integridade. Este processo possui uma norma específica, a *ISO/IEC 14764 – Software Maintenance* [ISO99], que é apresentada na seção 2.2.3.

Segundo [ROC01], os **Processos de Apoio** auxiliam e contribuem para o sucesso e a qualidade do projeto de software. Um processo de apoio é empregado e executado, quando necessário, por um dos seguintes processos:

- **Processo de documentação:** Define as atividades envolvidas no registro das informações do processo ou a atividade do ciclo de vida do software. Esse processo contém o conjunto de atividades que planejam, projetam, desenvolvem, produzem, editam, distribuem e mantêm aqueles documentos necessários a todos os interessados, tais como gerentes, engenheiros e usuários do sistema ou produto de software.
- **Processo de gerência de configuração:** Define as atividades para a aplicação de procedimentos administrativos e técnicos para todo o ciclo de vida do software, destinadas a identificar e definir os itens de software em um sistema e estabelecer suas

linhas básicas (*baselines*); controlar as modificações e liberações dos itens; registrar e apresentar a situação dos itens e dos pedidos de modificação; garantir a conclusão, a consistência e a correção dos itens; controlar o armazenamento, a manipulação e a distribuição dos itens de software.

- **Processo de garantia de qualidade:** Define as atividades para garantir a conformidade dos processos e produtos de software, no ciclo de vida do projeto, com seus requisitos especificados e sua aderência aos planos estabelecidos. A abrangência do processo inclui questões como garantia da qualidade do produto, do processo e do sistema de qualidade.
- **Processo de verificação:** Define as atividades para a verificação dos produtos de software. É um processo usado para determinar se os produtos de software de uma atividade atendem completamente aos requisitos ou às condições impostas a eles.
- **Processo de validação:** Define as atividades para a validação dos produtos produzidos pelo projeto de software. É usado para determinar se os requisitos e o produto final (sistema ou software) atendem ao uso específico proposto.
- **Processo de revisão conjunta:** Define as atividades para avaliar a situação e os produtos de uma atividade de um projeto. As revisões conjuntas são feitas tanto nos níveis de gerenciamento do projeto como nos níveis técnicos e são executadas durante a vigência do contrato.
- **Processo de auditoria:** Define as atividades para determinar a adequação do produto aos requisitos, aos planos e ao contrato, quando apropriado.
- **Processo de resolução de problemas:** Define um processo para analisar e resolver os problemas (incluindo não-conformidades), de qualquer natureza ou fonte, detectados durante o desenvolvimento, a operação, a manutenção ou a realização de outros processos. O objetivo é fornecer meios que garantam, em tempo adequado e de maneira responsável e documentada, a análise e a resolução de todos os problemas encontrados e a identificação das tendências de novas ocorrências.

De acordo com [ROC01], os **Processos Organizacionais** são empregados por uma organização para estabelecer e implementar uma estrutura constituída pelos processos do ciclo de vida e pelo pessoal envolvido no desenvolvimento do software. Eles são geralmente empregados fora do domínio de projetos e contratos específicos; entretanto, os ensinamentos desses projetos e contratos contribuem para a melhoria da organização. São eles:

- **Processo de gerência:** Define as atividades genéricas que podem ser empregadas por quaisquer das partes que têm de gerenciar seu(s) respectivo(s) processo(s). O gerente é responsável pelo gerenciamento do produto, do projeto e das tarefas do(s) processo(s) aplicável(eis) ao desenvolvimento do software, tais como aquisição, fornecimento, desenvolvimento, operação, manutenção e apoio.

- **Processo de infra-estrutura:** Define as atividades para estabelecer e manter a infra-estrutura necessária para qualquer outro processo. A infra-estrutura pode incluir hardware, software, ferramentas, técnicas, padrões e recursos para o desenvolvimento, a operação e a manutenção do software.
- **Processo de melhoria:** Define as atividades básicas que uma organização (isto é, adquirente, fornecedor, desenvolvedor, operador, mantenedor ou o gerente de outro processo) executa para estabelecer, avaliar, medir, controlar, e melhorar um processo de ciclo de vida de software.
- **Processo de treinamento:** Define as atividades para oferecer e manter pessoal treinado. A aquisição, o fornecimento, o desenvolvimento, a operação ou a manutenção de produtos de software é extremamente dependente de pessoal qualificado. Portanto, é essencial que o treinamento de pessoal seja planejado e implementado com antecedência para que o mesmo esteja disponível quando o produto de software for adquirido, fornecido, desenvolvido, operado ou mantido.

Para [ROC01], o **Processo de Adaptação** define as atividades necessárias para adaptar a norma para sua aplicação na organização ou em projeto. A adaptação deve ser executada com base em alguns fatores que diferenciam uma organização ou projeto de outros, dentre os quais a estratégia de aquisição, modelos de ciclo de vida de projeto, características de sistemas, software e cultura organizacional. A existência deste processo permite que a norma seja adaptável a qualquer projeto, organização, modelo de ciclo de vida, cultura ou técnica de desenvolvimento.

### 2.1.4 Rational Unified Process

Segundo [KRU00], RUP é um processo da engenharia de software que apresenta uma abordagem disciplinada para atribuir e organizar tarefas em uma organização de desenvolvimento de software. O objetivo deste processo é produzir software com alta qualidade, que atenda às necessidades do usuário, dentro do cronograma e orçamento previstos. Também é tratado pela empresa que o desenvolveu, a *Rational Software*, como um produto de software e, como tal, é mantido, atualizado e aperfeiçoado de acordo com as experiências da *Rational* e de seus usuários [RAT98]. Além disto, RUP representa um *framework* de processo genérico, podendo ser adaptado e estendido de acordo com as necessidades da organização. Pode ser aplicado a diferentes áreas, tipos de organização, níveis de competência e tamanhos de projeto [JAC01].

Nas seções a seguir serão apresentadas as principais características deste processo e o seu funcionamento será descrito tanto do ponto de vista estrutural quanto do ponto de vista comportamental.

### *2.1.4.1 Principais Características*

Para [JAC01] os três aspectos que representam o diferencial do RUP são: centrado na arquitetura, dirigido a casos de uso e iterativo e incremental. O aspecto iterativo e incremental do processo será abordado quando for apresentada a estrutura dinâmica do processo. As demais características serão apresentadas a seguir.

De acordo com [KRU00], RUP apresenta um grande foco em modelagem por ser centrado na arquitetura. A utilização de modelos ajuda a entender e a desenvolver o problema e a solução, pois representam uma simplificação da realidade. Para representar todos os aspectos do desenvolvimento de software é preciso utilizar múltiplos modelos com diferentes preocupações. Entretanto, é preciso assegurar que os modelos utilizados são consistentes e não apresentam redundâncias. Para que uma organização adote uma arquitetura, ela precisa atender às seguintes questões: entender a importância do uso da arquitetura e os benefícios de sua utilização; utilizar uma representação formal para definir a arquitetura; definir um processo arquitetural capaz de criar e validar uma arquitetura que satisfaça às necessidades do projeto.

De acordo com [KRU00], RUP é dirigido a casos de uso. Diversas abordagens apresentam o modelo do problema, que modela os requisitos e regras do sistema a ser desenvolvido. Para construir um modelo de caso de uso, primeiro é preciso entender os dois conceitos-chave relacionados: **caso de uso** é uma seqüência de ações que o sistema executa que produz um resultado observável para determinado ator; **ator** é alguém ou alguma coisa de fora do sistema que interage com o sistema (funções que pessoas ou outros sistemas podem assumir). Atores são entidades que interagem com o sistema e os casos de uso definem estas interações.

Como RUP é uma abordagem dirigida a casos de uso, isto significa que os casos de uso definidos para determinado sistema servem de base para todo o processo de desenvolvimento. O modelo de casos de uso é desenvolvido no *workflow* de requisitos e os casos de uso são utilizados para capturar a visão do sistema do ponto de vista do usuário. Desta forma, os casos de uso servem como uma linguagem comum para comunicação entre o usuário e a equipe responsável pelo projeto. Nos *workflows* de análise e projeto, casos de uso servem como ponte de ligação entre requisitos e atividades de projeto.

### *2.1.4.2 Estrutura Estática*

A estrutura estática apresenta os principais conceitos, ou elementos de modelagem, necessários para a compreensão do Processo Unificado. Conforme apresentado em [KRU00], RUP é representado por quatro elementos principais de modelagem:

- **Trabalhador:** define o comportamento e as responsabilidades de um indivíduo ou de um grupo trabalhando como um time. O comportamento é expresso como as atividades executadas pelo trabalhador, e cada trabalhador é associado a um conjunto coeso de atividades. As responsabilidades do trabalhador são

normalmente associadas a determinado artefato que ele cria, modifica ou controla, conforme ilustrado na Figura 1.

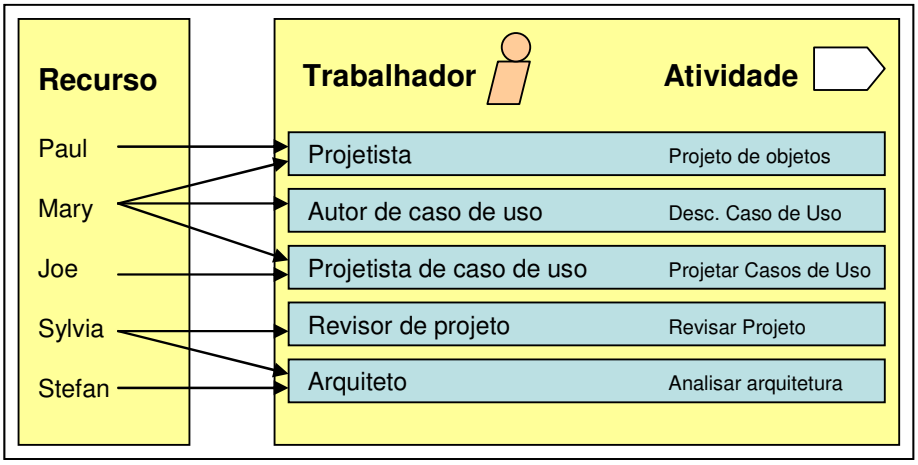


Figura 1 - Relação entre pessoas, trabalhadores e atividades (adaptado de [RAT98])

- **Atividades:** estão relacionadas a trabalhadores e definem o trabalho que eles deverão executar. Representam uma unidade de trabalho que deve ser relacionada à determinada função e deve produzir um resultado significativo no contexto do projeto. Além disto, a atividade possui um propósito bem definido, como criar ou atualizar um artefato, produzir um modelo, uma classe ou um plano. Cada atividade está usualmente relacionada a um trabalhador específico.
- **Artefatos:** representam as entradas e saídas das atividades. Um artefato é uma informação que é produzida, utilizada ou modificada pelo processo. Os artefatos podem ser representados de diversas formas: modelos; elementos de um modelo; documentos; código fonte; programa executável. Os artefatos produzidos em RUP são organizados em cinco conjuntos: gerência, requisitos, projeto, implementação e entrega.
- ***Workflow*:** é uma seqüência de atividades que produz um resultado com valor observável. Em UML pode ser representado pelo diagrama de colaboração, de seqüência ou de atividades. A mera enumeração de trabalhadores, atividades e artefatos não constituem um processo. É preciso descrever de maneira significativa a seqüência de atividades que produz algum resultado e apresentar a interação entre trabalhadores.

O *workflow* de núcleo representa o agrupamento de todos os trabalhadores e atividades de forma lógica. Existem seis *workflows* de engenharia (modelagem de negócios, requisitos, análise e projeto, implementação, teste e implantação) e três de suporte (gerência de projeto, configuração e gerência de mudanças e ambiente). Apesar dos nomes dos seis *workflows* de engenharia lembrar as fases seqüenciais do modelo em cascata tradicional, RUP representa um processo iterativo,

todos estes *workflows* são revisitados diversas vezes durante o ciclo de vida. O *workflow* real de um projeto intercala cada um dos nove *workflows* de núcleo repetindo-os com diferentes ênfases e intensidades em cada iteração. Uma descrição mais detalhada destes *workflows* pode ser encontrada em [JAC01], [KRU00] e [RAT98]. A Figura 2 ilustra a organização do dos *workflows* e a relação com a estrutura dinâmica do ciclo de desenvolvimento.

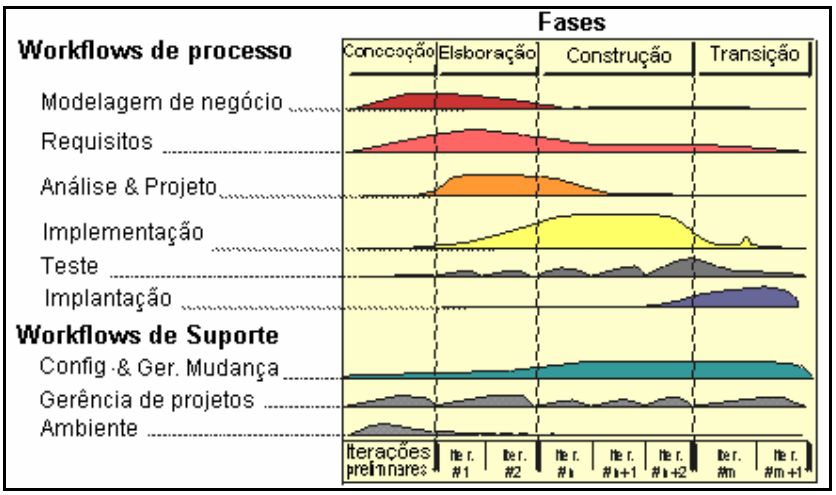


Figura 2 - Relação entre o *workflow* de núcleo e a estrutura dinâmica do ciclo de desenvolvimento [RAT98]

O *workflow* de detalhe, ilustrado na Figura 3, é utilizado para expressar um conjunto de atividades que estejam intimamente relacionadas. Também podem detalhar fluxos de informação (artefatos de entrada e saída das atividades), mostrando como as atividades atuam sobre os artefatos.

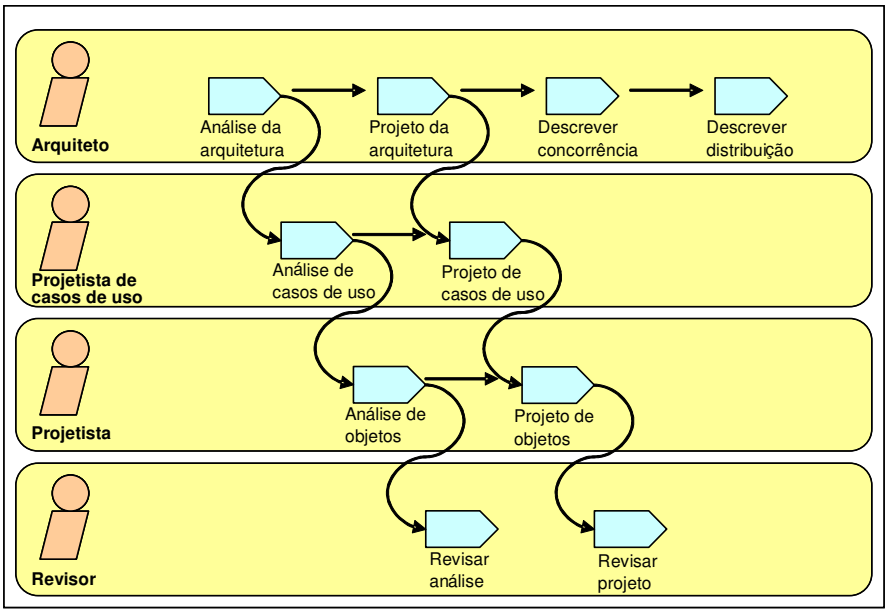


Figura 3 - Exemplo de *workflow* de detalhe [RAT98]

Os planos de iteração representam instâncias do processo para determinada iteração,

selecionando as atividades que deverão ser executadas durante a iteração e replicando-as quando necessário. Existem diversas formas de instanciar um processo e RUP apresenta alguns planos de iteração típicos.

Embora trabalhadores, atividades, artefatos e *workflows* representem os elementos fundamentais da estrutura estática de RUP, existem outros elementos que podem ser utilizados para facilitar o entendimento do processo e propiciar um guia para quem utiliza. Os elementos adicionais são: diretrizes (*guidelines*), *templates*, ferramentas de apoio e conceitos-chave. Pode-se encontrar uma descrição detalhada destes elementos em [KRU00].

### *2.1.4.3 Estrutura Dinâmica*

A estrutura dinâmica apresenta o RUP do ponto de vista comportamental e descreve como é organizado o ciclo de vida do processo. De acordo com [KRU00], o processo de desenvolvimento seqüencial (cascata) segue tipicamente cinco passos: completo entendimento do problema, dos seus requisitos e regras; projeto da solução que satisfaça às necessidades estipuladas; implementação da solução; verificação da implementação; entrega do produto. O que ocorre é que este é um processo válido para os casos em que o domínio do problema é bem conhecido e que os engenheiros envolvidos podem basear-se em centenas de anos de experiência. Como esta tipicamente não é a realidade da engenharia de software, RUP adota o conceito de desenvolvimento iterativo.

O **processo interativo** em RUP propõe quebrar o projeto numa sucessão de pequenos projetos seqüenciais. Desta forma, pode-se indicar alguns requisitos e riscos, projetar, implementar e validar este pedaço para então levantar mais alguns requisitos e recomeçar o processo.

Do ponto de vista da gerência de projeto, é preciso assegurar que o projeto esteja progredindo rumo ao produto final e que não fique passando de iteração em iteração sem propósito ou objetivo. Por isto, é preciso definir pontos no tempo (*milestones*) onde deve ser feita uma verificação do andamento do projeto para que se possa decidir continuar, abortar ou mudar o rumo do projeto (Figura 4).

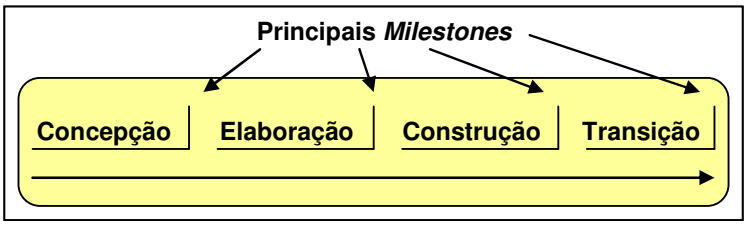


Figura 4 - Principais *Milestones* [RAT98]

Do ponto de vista de gerência, o processo é organizado em quatro fases principais (Figura 5): **concepção** (especificar uma visão do produto final e do negócio a fim de delimitar o escopo do projeto), **elaboração** (definir atividades e recursos necessários com o propósito de

especificar características e projetar a arquitetura), **construção** (construir o produto e permitir a evolução da visão, da arquitetura e do plano de projeto até finalizar o produto de acordo com a especificação do usuário) e **transição** (transição para a comunidade de usuários a qual implica manufatura, entrega, treinamento e manutenção do produto). A passagem pelas quatro fases é chamada **ciclo de desenvolvimento** e este ciclo produz a chamada **geração** do software. A menos que a vida útil do produto termine, o produto irá evoluir através da repetição da mesma seqüência de fases, produzindo uma nova geração para o software. Esta fase é chamada de **evolução**. Os ciclos de evolução podem ser disparados por diversos fatores como, por exemplo: aprimoramento, mudança de contexto, novas tecnologias, competição no mercado, entre outros.

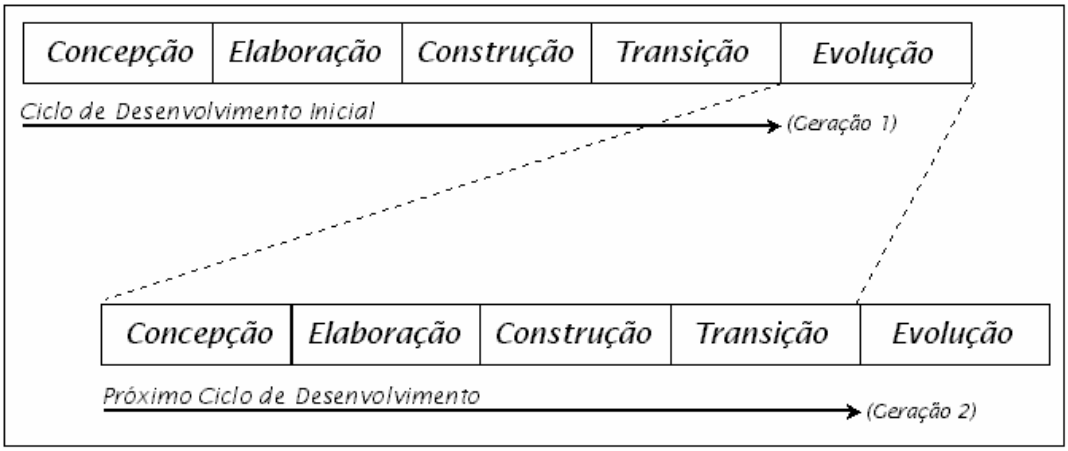


Figura 5 - Evolução do Ciclo de desenvolvimento (adaptado de [RAT96])

Do ponto de vista técnico, o desenvolvimento de software é visto como uma sucessão de iterações. Cada iteração é concluída pela liberação de um produto executável que pode representar um subconjunto da visão. A perspectiva de gerência e a perspectiva técnica são paralelas e, em geral, o fim de uma fase é sincronizado com o final das iterações. Ou seja, cada fase é quebrada em uma ou mais iterações.

### 2.1.5 Microsoft Solutions Framework

Apesar de o MSF não representar um padrão como a norma ISO/IEC 12207, nem tão pouco ser amplamente utilizado como o RUP, ele é utilizado na organização onde será desenvolvida a pesquisa deste autor durante o mestrado. Isto torna a sua apresentação no contexto do processo de software necessária, dada a influência que terá sobre os trabalhos futuros.

De acordo com [MIC04], MSF é um *framework* de processo flexível e adaptável que pode ser utilizado em projeto de diversos tamanhos ou complexidade, para planejar, construir e implantar soluções tecnológicas. O objetivo do MSF é fornecer soluções de TI de forma rápida, necessitando de poucas pessoas, envolvendo menos riscos e buscando resultados de alta

qualidade. A filosofia do MSF é que não existe uma única estrutura ou processo que se aplique igualmente aos requisitos e ambiente de todos os projetos. Entretanto, reconhece que existe a necessidade de orientação na busca por um processo adequado a cada caso. MSF fornece esta orientação sem impor detalhes prescritivos que tragam limitações ao conjunto de cenários onde o *framework* pode ser usado.

Segundo seus autores, os componentes do MSF podem ser usados individualmente ou coletivamente nos seguintes tipos de projetos:

- Projetos de desenvolvimento de software;
- Projetos de implantação de infra-estrutura;
- Projetos de integração de aplicações;
- Qualquer combinação dos tipos anteriores.

Nas seções a seguir serão apresentados os princípios fundamentais deste *framework* de processo e o seu funcionamento será descrito tanto do ponto de vista estrutural quanto do ponto de vista comportamental.

#### *2.1.5.1 Princípios Fundamentais*

Segundo [MIC04], os princípios fundamentais estabelecem valores e padrões presentes em todos os elementos do *framework*. O MSF é baseado em oito princípios fundamentais:

- Promova Comunicações Abertas;
- Trabalhe por uma visão compartilhada;
- Potencialize os membros da equipe;
- Estabeleça claramente as responsabilidades compartilhadas e as formas de avaliação;
- Tenha foco em agregar valor ao negócio;
- Seja ágil, esteja preparado para mudanças;
- Invista em qualidade;
- Aprenda com todas as experiências.

Juntos, estes princípios expressam a filosofia do MSF e formam a base para sua organização de pessoas e processos. Permeiam tanto a estrutura quanto a aplicação do MSF.

#### *2.1.5.2 Estrutura Estática*

A estrutura estática do MSF é representada pelo seu modelo de equipe. De acordo com [MIC04], o modelo de equipe define os papéis e as responsabilidades em times que trabalham em projetos de TI. A Figura 6 apresenta a organização lógica do modelo de equipe.

O modelo de equipes do MSF é baseado na premissa de que qualquer projeto de tecnologia deve atingir certos objetivos chave de qualidade para ser considerado bem sucedido. Atingir estes objetivos requer a aplicação de um conjunto de habilidades e áreas de conhecimento relacionadas, sendo que cada uma delas é agrupada em papéis da equipe. As habilidades e áreas de

conhecimento relacionadas são chamadas de áreas funcionais e definem o domínio de cada papel da equipe. Coletivamente, estes papéis têm a responsabilidade de deliberar a respeito de todos os critérios de sucesso do projeto. Mas cada papel é considerado de igual importância e as decisões importantes devem ser tomadas em conjunto, com cada papel colaborando com a perspectiva única de sua incumbência. Os objetivos associados a cada papel são apresentados na Tabela 1.

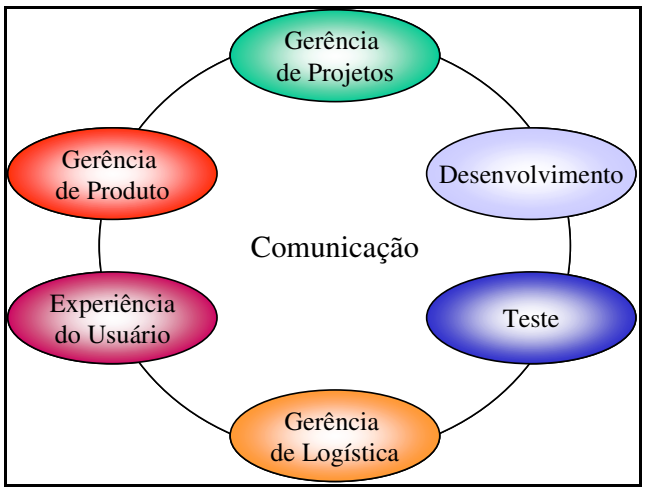


Figura 6 - Modelo de Equipe do MSF (adaptado de [MIC04])

Tabela 1 – O modelo de equipe do MSF e os objetivos chave de qualidade (adaptado de [MIC04])

| Objetivo Chave de Qualidade | Papel na equipe MSF |
|---|---|
| Término do projeto dentro das restrições estabelecidas | Gerência de Projetos |
| Término do projeto respeitando as especificações | Desenvolvimento |
| Liberação do produto somente depois de tratadas todas as falhas | Teste |
| Implantação bem sucedida do produto | Gerência de Logística |
| Performance do usuário melhorada | Experiência do Usuário |
| Satisfação do usuário | Gerência de Produto |

É necessário ressaltar que um papel não representa necessariamente uma pessoa. Várias pessoas podem responder por um papel, ou uma única pessoa pode responder por diversos papéis, dependendo do tamanho e das necessidades do projeto. O importante é que todos os objetivos sejam representados no projeto e que todos os participantes tenham conhecimento de quem no projeto é o responsável por cada um deles.

Os papéis apresentados são aplicados em conjunto com os processos, que serão apresentados na próxima seção, para descrever as atividades e os artefatos que devem ser produzidos por elas.

### *2.1.5.3 Estrutura Dinâmica*

A estrutura dinâmica apresenta o MSF do ponto de vista comportamental e descreve como é organizado o ciclo de vida do *framework*. De acordo com [MIC04], cada projeto desenvolve-se através de um ciclo de vida. No MSF este ciclo de vida é representado como um modelo de processo, um processo que inclui todas as atividades necessárias para atingir-se um estado operacional do produto. A principal função deste modelo de processo é estabelecer a ordem em que as atividades do projeto devem ser realizadas. Uma visão simplificada do modelo de processo do MSF é apresentada na Figura 7.

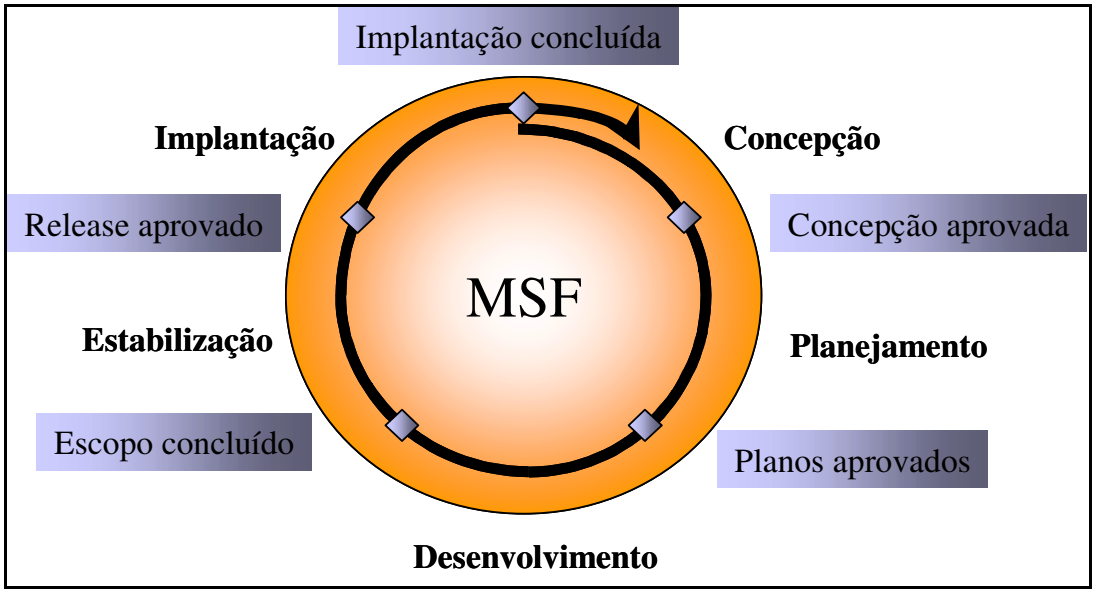


Figura 7 - Modelo de Processo do MSF (adaptado de [MIC04])

De acordo com [MIC04], o objetivo dos autores do MSF ao criar este modelo de processo foi combinar conceitos dos modelos de ciclo de vida em cascata e em espiral para extrair as vantagens de cada um deles. O modelo de processo combina os benefícios do planejamento baseado em *milestones* do modelo em cascata, com as entregas iterativas do modelo em espiral.

O modelo é baseado em fases e *milestones*. As fases podem ser consideradas como um período no tempo com determinada ênfase em determinadas atividades. Entretanto, cada fase tem suas próprias características e o final de cada fase representa uma mudança no foco do projeto.

Já os *milestones* são pontos de revisão e sincronização para determinar se os objetivos de cada fase foram atingidos. Os *milestones* também oferecem oportunidades para que o time ajuste o escopo do projeto para adequá-lo às mudanças ocorridas nos requisitos do projeto e para acomodar riscos e problemas surgidos durante a evolução do projeto.

## 2.2 Manutenção de Software

A manutenção de software é a atividade que demanda o maior volume de esforço dentre todas as atividades de engenharia de software. Muitos dos sistemas dos quais as organizações dependem atualmente foram desenvolvidos há mais de uma década. E mesmo que estes sistemas tenham sido desenvolvidos usando-se as melhores práticas de projeto e codificação disponíveis naquela época (e muitos não foram), as principais preocupações eram com o tamanho dos programas e o espaço de armazenamento que exigiriam. Desde então, estes sistemas foram migrados para novas plataformas, ajustados devido a mudanças nos equipamentos e nos sistemas operacionais e melhorados para atender novos requisitos funcionais. O resultado são aplicações mal estruturadas, mal codificadas e fracamente documentadas [PRE01].

Fatores como estes levaram a manutenção de software a ser comparada a um iceberg onde a maior quantidade de problemas e custos escondem-se sob a superfície, pois não são facilmente identificáveis.

Isto faz com que a manutenção de software consuma a maior parte de todo o esforço despendido por uma organização de desenvolvimento [HAN93]. Desde então, esta demanda tem aumentado à medida que mais produtos de software são desenvolvidos.

Esta seção apresenta o referencial teórico sobre a atividade de manutenção de software. São apresentados os principais conceitos ligados a manutenção de software e aos sistemas legados, a norma *ISO/IEC 14764 – Software Maintenance*, específica para o processo de manutenção de software, bem como algumas das abordagens sobre manutenção de software presentes na literatura.

### 2.2.1 Conceitos

A manutenção de software é definida por [IEE98] como a modificação de um produto de software depois de sua entrega (ao cliente) para corrigir erros, melhorar sua performance ou qualquer outro atributo, ou para adaptar o produto a um ambiente modificado. Esta definição é muito semelhante àquela apresentada por [MAR83] que definiu a manutenção de software como todo o trabalho realizado em um produto de software após este ser colocado em produção, incluindo correções, alterações, melhorias e otimizações.

Este processo normalmente é desencadeado por uma solicitação do cliente ou por algum relatório de problemas gerado pelo usuário, sendo que este tipo de solicitação é identificada pelo termo genérico de Requisição de Modificação ([IEE98] e [ISO99]).

#### *2.2.1.1 Classificação*

Os principais autores da área de manutenção de software [IEE98][ISO99][PRE01][SOU98] [SNE03] convergem para uma tipologia básica para a manutenção de software:

- **Corretiva:** Modificação reativa de um produto de software realizada para corrigir falhas descobertas após sua entrega [IEE98][ISO99][PRE01][SOU98]. Visa colocar o software no estado em que ele deveria estar desde que este estado tenha sido definido [SNE03];
- **Emergencial:** Manutenção corretiva não planejada, feita em caráter emergencial para manter o software operacional [IEE98];
- **Adaptativa:** Modificação realizada em um produto de software, após sua entrega, para manter o programa funcionando em um ambiente diferente ou que esteja mudando [IEE98][ISO99][PRE01]. Para [SOU98] inclui todo o trabalho iniciado como conseqüência da migração de um software existente para uma nova plataforma de hardware ou software. Entretanto, para [SNE03] a manutenção adaptativa modifica funcionalidades para inserir novos requisitos no lugar de requisitos já existentes;
- **Perfectiva ou de Melhoria:** Para [SNE03] e [SOU98] tanto podem ser manutenções que visam aperfeiçoar características não-funcionais, como performance ou facilidade de manutenção de um software, quanto acrescentar novas funcionalidades. Já para [IEE98] e [ISO99] são apenas melhorias de performance ou facilidade de manutenção, enquanto que para [PRE01] são apenas acréscimos de funcionalidades;
- **Preventiva:** Sistemas de software tendem a se deteriorar a medida que são modificados e, em virtude desta tendência, a manutenção preventiva, também chamada de reengenharia, deve ser empregada para garantir que o software continue servindo a seus propósitos. Em essência, a manutenção preventiva modifica os sistemas para torná-los mais facilmente corrigíveis, adaptáveis ou aperfeiçoáveis [PRE01][SOU98]. Adicionalmente, em [ISO99], a manutenção preventiva é definida como uma modificação em um produto de software, após sua entrega ao cliente, para detectar e corrigir falhas latentes antes que elas se tornem falhas efetivas.

Entretanto, [ISO99] sugere uma hierarquia, entre estes diferentes tipos de manutenção, que classifica as manutenções adaptativas e perfectivas como tipos de melhoria e as manutenções corretivas e preventivas como tipos de correções, conforme apresentado na Figura 8.

Adicionalmente aos conceitos apresentados na classificação dos tipos de manutenção, no caso de manutenção de melhoria, de acordo com [SNE03], deve haver um limite no tamanho da melhoria para distingui-la de um novo desenvolvimento reusando porções do sistema existente. Muitas companhias estabelecem políticas limitando a taxa de crescimento anual em 25% para continuar considerando que o produto está em manutenção. Se esta taxa é excedida o produto

está na fronteira de se tornar outro produto e o projeto deverá ser considerado como um projeto de desenvolvimento. Esta política estabelece um limite superior para as manutenções de melhoria em termos de acréscimo de funcionalidades.

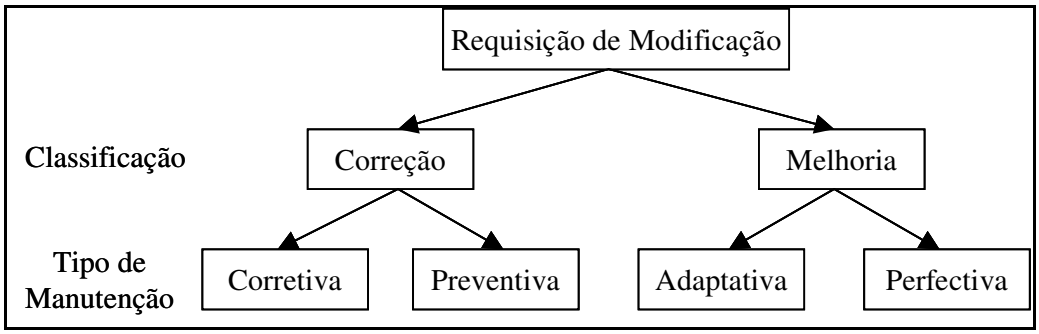


Figura 8 - Classificação dos tipos de manutenção (adaptado de [ISO99])

As diferentes formas de manutenção ainda podem ser analisadas do ponto de vista econômico. As manutenções de melhoria podem agregar valor ao produto, enquanto que nas manutenções corretivas e adaptativas o valor do produto permanece inalterado. Mas é preciso distinguir o tipo de melhoria, pois apesar de um incremento de funcionalidade acarretar também um incremento no preço do produto, uma melhoria na facilidade de manutenção ou na performance do software deverá acarretar apenas um acréscimo nos custos do projeto [SNE03].

#### *2.2.1.2 A Manutenção no Processo de Software*

Uma característica comum às definições de manutenção é aquela que estabelece que este processo deva ocorrer após a entrega do produto ao cliente. Esta característica parece sugerir uma relação de ordem no tempo que coloca o processo de manutenção logo após o processo de desenvolvimento do produto. Mas não estabelece a necessidade de ter havido o término do processo de desenvolvimento antes do início das atividades de manutenção.

De acordo com [ISO98], o custo da manutenção e a facilidade dos mantenedores em conduzir as operações de manutenção são fortemente influenciados pelo que ocorre ou deixa de ocorrer durante o processo de desenvolvimento do software. Em muitos casos os mantenedores não têm a oportunidade de participar do processo de desenvolvimento, entretanto deveriam participar sempre que possível. As funções que o mantenedor pode realizar durante o processo de desenvolvimento incluem:

- Planejamento para a logística de suporte ao produto de software;
- Garantir que o produto será suportável;
- Dar suporte ao planejamento da transição do produto da fase de desenvolvimento para a fase de manutenção.

A capacidade de um software de ser mantido (*maintainability*) também precisa ser garantida durante o desenvolvimento do software. É necessário que o desenvolvedor estabeleça planos,

práticas, recursos e atividades relativas a este objetivo.

Além disto, a manutenção é necessária independentemente do modelo de ciclo de vida empregado (por exemplo: cascata, incremental, etc.). Portanto, no caso de ciclos incrementais, mesmo que o processo de desenvolvimento continue após a entrega de um release, pode ser necessário iniciar o processo de manutenção para o release já implantado. Isto implica na coexistência dos dois processos para o mesmo produto e contradiz a aparente relação de ordem no tempo entre estas fases do ciclo de vida do software.

### 2.2.2 Sistemas Legados

Como foi visto, a manutenção de software pode ser necessária mesmo em sistemas que ainda estão em desenvolvimento ou que acabaram de ser entregues ao seu usuário. Entretanto, é a atividade de manutenção em sistemas legados que apresenta os maiores desafios. Esta secção apresenta a definição de sistema legado e sua classificação.

#### *2.2.2.1 Definição*

É possível observar na literatura uma convergência quanto às características dos sistemas legados. Alguns autores usam o termo "sistemas legados" [BIA03][LUC01][STE98], outros o termo "programa legado" [PRE01] ou ainda "aplicação legada" [UMA97]. Mas para a maioria a definição está relacionada à quando o sistema foi desenvolvido, qual sua importância para a organização e as várias manutenções as quais já foi submetido.

Para [UMA97] uma aplicação legada é uma aplicação valiosa, ou seja, crítica para o negócio, herdada do passado. Para [LUC01] é um sistema de software de missão crítica, desenvolvido em algum momento do passado, que ainda é usado e vem sendo modificado ao longo do tempo sem submeter-se a ações sistemáticas de melhoria. Para [STE98] são aqueles sistemas que possuem valor para a organização e ainda resistem às modificações e a evolução para adequar-se a requisitos de negócio em constante mudança. E para [PRE01] um programa legado é um sistema antigo, freqüentemente mal projetado ou documentado, mas que é critico para o negócio e deve ser suportado por vários anos. Portanto, os principais atributos de um sistema legado são: sua idade, sua criticidade e a quantidade de manutenções as quais o sistema já foi submetido.

É importante também destacar que [BRO95] define um sistema crítico como sendo essencial para a continuidade dos negócios da organização e, portanto, o sistema deve estar permanentemente operacional. O risco envolvido na substituição de um sistema crítico para o negócio é alto, o que leva a organização a optar por sua manutenção. Isto explica porque sistemas voltados a atividades meio não se tornam legados, enquanto que sistemas voltados a atividades fim tendem a se tornarem sistemas legados.

#### *2.2.2.2 Categorias de Sistemas Legados*

Os sistemas legados podem ser vistos como um conjunto de dados legados, funcionalidades e interface de usuário. Esta visão tem origem na arquitetura de tais sistemas, e permite analisar a relação entre suas camadas, conforme apresentado na Figura 9. De acordo [UMA97] as aplicações legadas podem ser classificadas como:

- **Aplicações Altamente Decomponíveis:** São muito bem estruturadas. Apresentam componentes de aplicação separados em processamento de interface de usuário, lógica de aplicação e serviços de acesso a dados. Os módulos da aplicação são independentes entre si e não existem interdependências hierárquicas. Em outras palavras são aplicações estruturadas em três camadas;
- **Aplicações de Dados Decomponíveis:** São aplicações semi-estruturadas ou semi-decomponíveis. Apresentam componentes separados em duas unidades: uma de acesso aos dados e outra onde a interface e a lógica de aplicação estão misturadas. Em outras palavras, são aplicações estruturadas em duas camadas;

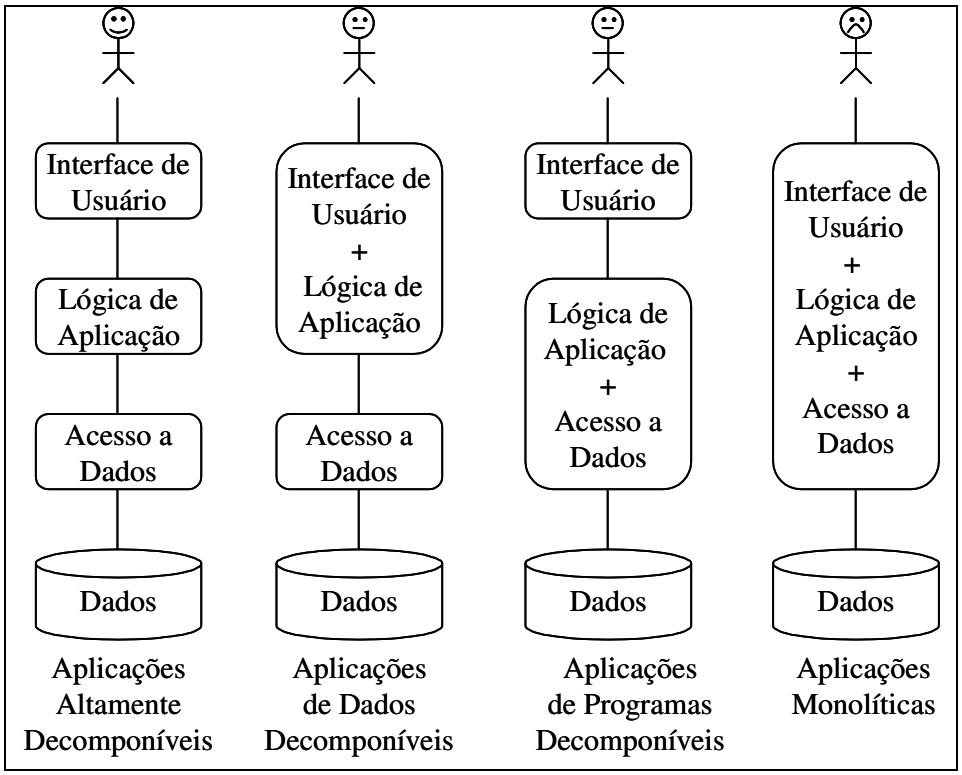


Figura 9 - Visão arquitetural das categorias de aplicações legadas.

- **Aplicações de Programas Decomponíveis:** Também são aplicações semi-estruturadas e que apresentam componentes separados em duas unidades. Entretanto, com uma distinção: um componente de processamento da interface e outro de lógica de aplicação altamente integrado com os serviços de acesso aos

dados. Os dados não podem ser diretamente acessados. Só estão disponíveis através da execução das funcionalidades do sistema;

- **Aplicações Monolíticas:** Aplicações não estruturadas. Todos os componentes da aplicação são vistos com uma unidade e a interface de usuário, a lógica de aplicação e os serviços de acesso a dados estão completamente misturados.

De forma geral as aplicações mais decomponíveis são mais "amigáveis" do ponto de vista da reengenharia, enquanto que as menos decomponíveis tornam-se mais "hostis".

### 2.2.3 A norma ISO/IEC 14764

Esta seção apresenta a norma *ISO/IEC 14764 – Software Maintenance* que descreve o processo de manutenção de software. Este processo fundamental do ciclo de vida de software, definido em [ISO98], contém as atividades e tarefas necessárias para modificar um produto de software existente preservando sua integridade [ISO99]. A Figura 10 apresenta as atividades deste processo.

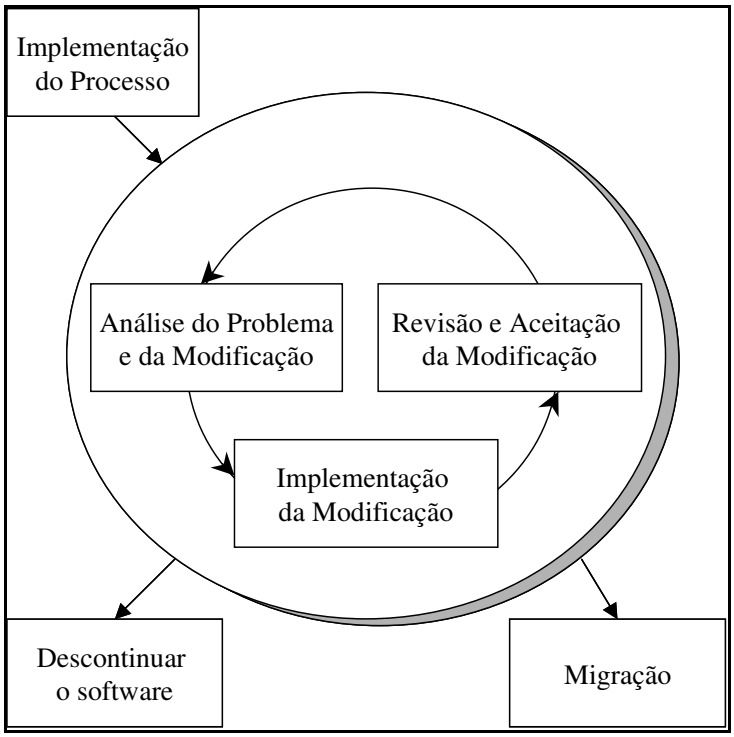


Figura 10 - Processo de manutenção de software (adaptado de [ISO99])

De acordo com [ISO99], estas atividades e tarefas são de responsabilidade do mantenedor de software, conforme estabelecido em [ISO98] e apresentado na seção 2.1.3. Cabe ao mantenedor certificar-se de que o processo de manutenção de software existe e é funcional antes de iniciar-se qualquer desenvolvimento de software.

Após a entrega do produto de software, o mantenedor deve modificar o código e a

documentação associada em resposta às requisições de modificação ou relatórios de problemas. Este processo oferece suporte ao produto desde sua concepção, sua migração para novos ambientes e até o software ser descontinuado. Deve ser ativado sempre que existir um requisito de que o produto de software deve ser mantido. Assim que o processo for ativado, planos e procedimentos devem ser desenvolvidos e recursos devem ser alocados especificamente para a manutenção. O processo só é encerrado quando o produto for finalmente descontinuado ou migrado. Nas sessões a seguir serão descritas as atividades que compõem este processo.

#### *2.2.3.1 Implementação do Processo*

Nesta atividade deve-se desenvolver, documentar e executar planos e procedimentos para conduzir as tarefas relativas ao processo de manutenção.

Em [ISO99] são relacionadas como necessárias para esta atividade as seguintes entradas:

- A *baseline* atual;
- A documentação do sistema;
- Uma requisição de modificação ou um relatório de problemas.

Para desenvolver e documentar a estratégia que será usada para conduzir o processo de manutenção, as seguintes tarefas devem ser executadas:

- Desenvolver os Planos e Procedimentos de Manutenção;
- Estabelecer procedimentos para tratar as Requisições de Modificação e os Relatórios de Problemas;
- Implementar o Processo de Gerência de Configuração.

O plano de manutenção deve ser desenvolvido em paralelo com o plano de desenvolvimento. Deve-se estabelecer procedimentos para receber, armazenar, acompanhar os pedidos de modificação e os relatórios de problemas dos usuários e fornecer um retorno para os mesmos. Além disto, uma interface organizacional com o processo de gerência de configuração também deve ser feita para que as modificações realizadas no sistema ou produto de software possam ser gerenciadas.

São geradas como saídas desta atividade:

- Plano de Manutenção;
- Procedimentos de manutenção;
- Procedimentos para resolução de problemas;
- Planos para o retorno do usuário;
- Plano de Transição;
- Plano de Gerência de Configuração.

#### *2.2.3.2 Análise do Problema e da Modificação*

Antes de modificar o produto o mantenedor deve analisar a Requisição de Modificação ou

o Relatório de Problema para avaliar o impacto que a manutenção terá no sistema existente, na organização e nos sistemas que com os quais o produto possui interfaces. Nesta análise o mantenedor também deve desenvolver e documentar as possíveis soluções e obter a aprovação necessária para iniciar a implementação da solução escolhida.

Esta análise deve considerar os seguintes aspectos:

- **O tipo:** Deve-se verificar se a manutenção é corretiva, adaptativa, perfectiva ou preventiva.
- **O escopo:** Deve-se verificar o tamanho, os custos envolvidos e o esforço necessário para efetuar a manutenção.
- **A criticidade:** Deve-se verificar a importância da manutenção e seu impacto em aspectos como segurança e performance do sistema.

Devem ser realizadas as seguintes tarefas:

- Analisar as Requisições de Modificação e os Relatórios de Problemas;
- Verificar o problema;
- Desenvolver opções para implementar a solução;
- Documentar a análise das Requisições de Modificação e dos Relatórios de Problemas, os resultados e as opções de implementação;
- Obter aprovação para a manutenção.

São necessárias para esta atividade as seguintes entradas:

- A *baseline* atual;
- A documentação do sistema;
- Uma requisição de modificação ou um relatório de problemas;
- O repositório do software;
- A documentação do sistema.

Sendo que a documentação do sistema inclui:

- Informações sobre a configuração atual;
- Requisitos funcionais;
- Requisitos de interface;
- Dados sobre o planejamento do projeto;
- As saídas geradas pela atividade de implementação do processo.

São geradas como saídas desta atividade:

- Análise de Impacto;
- Opções recomendadas;
- Modificações aprovadas;
- Documentação atualizada.

Sendo que a análise de impacto deve incluir as seguintes informações:

- Definição do problema ou dos novos requisitos;
- Avaliação do problema ou dos requisitos;
- Classificação da manutenção que é necessária;
- Prioridades;
- Dados para verificação em caso de manutenção corretiva;
- Estimativa inicial dos recursos necessários para realizar a alteração do sistema existente.

Ao final desta atividade uma análise de riscos deve ser realizada. Usando as saídas desta atividade as estimativas iniciais devem ser reavaliadas e uma decisão deve ser tomada sobre prosseguir ou não para a atividade de implementação da modificação.

#### *2.2.3.3 Implementação da Modificação*

Durante a atividade de implementação da modificação o mantenedor deve desenvolver e testar a modificação que foi especificada e aprovada na atividade de análise do problema e da modificação.

São necessárias para esta atividade as seguintes entradas:

- A *baseline*;
- A Requisição de Modificação aprovada;
- A documentação atualizada.

A *baseline* deve incluir:

- Definições da arquitetura do sistema;
- Os registros das requisições de modificação;
- O código fonte;

O mantenedor deverá analisar a modificação solicitada e determinar quais unidades de software, programas e documentos devem ser alterados e a seguir evocar o mesmo processo de desenvolvimento definido na norma ISO/IEC 12207 e apresentado na seção 2.1.3. Devem ser realizadas as mesmas atividades e empregadas as mesmas técnicas de engenharia de software que são usadas em projetos de desenvolvimento de software para a atividade de implementação do software. Sendo que estas atividades devem ser ajustadas para as necessidades de manutenção de software sempre que necessário.

São geradas como saídas desta atividade:

- Planos e procedimentos de teste atualizados;
- Documentação atualizada;
- Código fonte atualizado;
- Relatório de teste;

- Métricas.

Sendo que a documentação atualizada deve incluir:

- Registros de modificação atualizados;
- Relatório detalhado de análise;
- Requisitos atualizados;
- Material de treinamento atualizado.

#### *2.2.3.4 Revisão e Aceitação da Modificação*

Esta atividade visa garantir que as modificações no sistema estão corretas e foram realizadas em conformidade com os padrões estabelecidos usando as metodologias corretas. O mantenedor deve conduzir a revisão com a organização que autorizou a modificação para averiguar a integridade do sistema modificado. Em seguida o mantenedor deve obter a aprovação formal de que a modificação foi realizada como especificado no contrato. Mesmo quando a manutenção for conduzida sem um contrato, a aprovação deve ser obtida.

As entradas necessárias para esta atividade são as seguintes:

- O software modificado;
- Resultados dos testes da modificação;

São geradas como saídas desta atividade:

- Nova *baseline*, incorporando as modificações realizadas;
- Modificações rejeitadas;
- Relatório de aprovação;
- Relatórios de auditoria e revisão;
- Relatório de testes para qualificação do produto.

#### *2.2.3.5 Migração*

Durante existência de um sistema, modificações podem tornar-se necessárias para poder rodar o software em um ambiente diferente. Para realizar esta migração o mantenedor deverá determinar, desenvolver e documentar as ações necessárias.

São necessárias para esta atividade as seguintes entradas:

- O ambiente antigo;
- O novo ambiente;
- A *baseline* antiga;
- A nova *baseline*;

Para realizar a migração em conformidade com esta norma o mantenedor deve planejar a migração, avaliar o impacto da migração, notificar os usuários a respeito da migração, fornecer-lhes treinamento e notificá-los do término da migração, além de arquivar todas as informações a respeito da migração.

São geradas como saídas desta atividade:

- Plano de migração;
- Ferramentas de migração;
- Notificações aos usuários;
- Produto de software migrado.

#### *2.2.3.6 Descontinuar o software*

Quando um software atinge o final de sua vida útil, ele deve ser descontinuado. Uma análise deve ser realizada para apoiar a decisão de descontinuar ou não um software. Esta análise é freqüentemente econômica e pode ser incluída no plano de descontinuidade. O software pode ser substituído por um novo software, mas em muitos casos não é substituído. Mesmo assim, para descontinuá-lo o mantenedor deve identificar, desenvolver e documentar as ações necessárias.

São necessárias para esta atividade as seguintes entradas:

- O produto de software antigo que será descontinuado;
- O novo produto de software;
- O ambiente antigo.

Para descontinuar um produto de software em conformidade com esta norma, o mantenedor deve desenvolver um plano de descontinuidade, notificar os usuários a respeito da descontinuidade, fornecer-lhes treinamento e notificá-los do término da descontinuidade, além de arquivar todas as informações a respeito da mesma.

São geradas como saídas desta atividade:

- Plano de Descontinuidade;
- Notificações aos usuários;
- Resultados da atividade;
- *Baseline* arquivada.

### 2.2.4 A manutenção de Software no RUP e no MSF

Como foi visto, a manutenção de software é um importante processo do ciclo de vida de um produto de software. Sendo incluído, por exemplo, na categoria de processo fundamental no padrão ISO/IEC 12207, apresentado na seção 2.1.3. Entretanto, os processos RUP e MSF, apresentados neste trabalho, não apresentam tratamentos consistentes para as atividades de manutenção de software.

No caso do RUP, a manutenção é tratada como uma nova fase no ciclo de vida do produto, chamada de evolução, que se caracteriza pela execução de um novo ciclo de desenvolvimento, após a entrega do produto ao cliente. Segundo [JAC01], RUP é proposto como um processo genérico, que pode ser adaptado e estendido de acordo com as necessidades da organização.

Portanto, para utilização em projetos de manutenção de software, torna-se necessária a configuração prévia do processo através da realização das atividades previstas no *workflow* de ambiente, que faz parte do RUP. Mas RUP não especifica claramente como as atividades de manutenção devem ser diferenciadas das atividades realizadas no desenvolvimento de um produto novo.

Já no caso do MSF não há nenhuma referência sobre como tratar a manutenção de software. E mesmo sendo definido como um *framework* de processo flexível e adaptável que pode ser utilizado em projeto de diversos tamanhos ou complexidade, não há nenhuma descrição clara sobre como o processo deve ser configurado para uso em projetos de manutenção, pois, segundo [MIC04], o MSF não impõe detalhes prescritivos que tragam limitações ao conjunto de cenários onde o *framework* pode ser usado.

Por fim, é importante destacar que, como foi visto na seção 2.2.3, muitos aspectos ligados à manutenção de software devem ser tratados durante a fase de desenvolvimento do produto. E que a manutenção do produto de software pode ser iniciada ainda durante a sua fase de desenvolvimento. Mas tanto no RUP quanto no MSF, existem poucas referências sobre como lidar com estes aspectos durante a fase de desenvolvimento do produto. As poucas referências restringem-se a considerar a facilidade de manutenção como mais um requisito não funcional do produto. Não há nenhuma referência quanto à participação do mantenedor durante o projeto de desenvolvimento e tão pouco quanto à realização de atividades inerentes a um processo de manutenção.

### 2.2.5 Aspectos Gerenciais da Manutenção de Software

Os sistemas legados atualmente são uma preocupação dominante entre gerentes de TI. De acordo com [UMA97], os gerentes de TI gostariam de não ter que lidar com nenhuma mudança em tais sistemas e várias são as razões para suas preocupações:

- Sistemas legados fornecem serviços vitais muito arriscados para serem parados;
- Os usuários e as equipes de suporte já estão treinados no uso destes sistemas;
- Muitos dos sistemas legados são confiáveis e realizam muito bem suas atividades, ao contrário da crença comum;
- As atividades administrativas e de suporte foram maturadas durante muitos anos;
- Existe certo apego emocional entre os grupos mais antigos da organização pelo fato de tais sistemas já terem sobrevivido a anos de mudanças fundamentais nas práticas de negócios e nas tecnologias.

Entretanto, de acordo com [UMA97], algo precisa ser feito a respeito de tais sistemas, pois:

- As aplicações legadas tornam-se cada vez mais dispendiosas para se manter e

operar;

- Estas aplicações não satisfazem as necessidades de flexibilidade e crescimento das modernas organizações;
- Muitos pacotes prontos estão se tornando populares, especialmente entre as pequenas e médias empresas, evidenciando as fraquezas dos sistemas legados;
- Muitos funcionários não desejam mais trabalhar com sistemas que foram criados antes de eles nascerem.

Diversas abordagens vêm sendo sugeridas para lidar com os aspectos gerenciais da manutenção de software. Esta seção apresenta algumas destas abordagens selecionadas na literatura.

#### *2.2.5.1 Fatores Críticos de Sucesso na Manutenção de Software*

Projetos de software costumam considerar sucesso como a produção de software com alta qualidade, que atenda às necessidades do usuário, dentro do cronograma e orçamento previstos. Mesmo em projetos de manutenção estes três fatores são importantes. Entretanto, [SNE03] considera que, apesar de estes serem bons pontos de partida, são apenas simplificações excessivas de um problema muito complexo.

Neste contexto, [SNE03] apresenta um estudo que enumera oito Fatores Críticos de Sucesso (FCS) visando colaborar com a mensuração do sucesso na manutenção de software em termos de objetivos quantificáveis. Os FCS identificados são:

- **Funcionalidade**: A operação de manutenção deve pelo menos preservar, senão melhorar, a funcionalidade do sistema em manutenção. O usuário não deve receber, após a manutenção, menos funcionalidades do que tinha antes dela. O que significa que cada novo release deve ter, pelo menos, as mesmas funcionalidades e dados presentes nos *releases* anteriores, enquanto elas ainda forem necessárias. Uma funcionalidade ou um dado só pode ser excluído com a concordância de todos os usuários e somente se esta modificação não acarretar em efeitos colaterais indesejados. Uma perda de funcionalidade ou informação pode ser considerada como uma erosão das funcionalidades do sistema;
- **Qualidade**: A operação de manutenção deve preservar, senão melhorar, a qualidade do sistema em manutenção. Um importante objetivo de qualquer projeto é melhorar a qualidade do produto através da eliminação preventiva das causa de erros, ajustando o sistema para usar menos recursos ou melhorando a qualidade do código e da documentação. Entretanto é necessário ter-se o cuidado de não permitir que os custos ultrapassem os limites orçamentários do projeto. A qualidade geral do produto é determinada durante a fase de desenvolvimento. Após a entrega, e assim que o produto entra em manutenção, torna-se muito custoso e

arriscado tentar elevar significativamente a qualidade do produto, a menos que haja uma solicitação explícita do cliente para fazê-lo;

- **Complexidade**: A operação de manutenção não deve aumentar a complexidade do produto relativa ao seu tamanho. Se por um lado a operação de manutenção visa preservar a funcionalidade e a qualidade, por outro lado ela deve controlar a complexidade do software. Um sistema grande tem diferentes complexidades em diferentes níveis. No nível de componentes, no nível de subsistemas e no nível de sistema. Uma operação de manutenção não deve levar a um acréscimo no número de componentes e de subsistemas, bem como não deve aumentar o nível de interação entre os componentes e entre os subsistemas. Deve ao menos manter a complexidade no mesmo nível, senão reduzi-la como resultado de esforços de reengenharia;
- **Volatilidade**: A operação de manutenção não deve levar a um aumento na volatilidade do produto. A volatilidade de um sistema é definida como a propensão de um sistema a mudar seu estado de uma evolução para uma revolução, sendo que o autor define como revolução um estado em que o sistema demanda uma revisão geral e um novo ciclo completo de desenvolvimento. Esta propensão pode ser medida pela quantidade e extensão das requisições de alteração solicitando melhorias comparadas com a quantidade das demais tarefas de manutenção. Melhorias causam profundas mudanças estruturais e devem diminuir à medida que o sistema evolui. Caso elas voltem a crescer, será um sinal de que o sistema logo não atenderá mais aos requisitos e às necessidades do usuário. O crescimento da volatilidade é sinal de riscos extremamente altos;
- **Custo**: O custo relativo por tarefa de manutenção não deve crescer, desde que provada a similaridade de escopo entre as tarefas. O custo médio de realizar-se uma manutenção, relativo ao tamanho do impacto é outro importante fator de sucesso. O impacto de uma alteração pode ser definido em termos de pontos por função, pontos por objeto, instruções ou qualquer outra métrica de tamanho. O esforço em homens-dia em relação à métrica de tamanho fornece a taxa de produtividade. A produtividade em projetos de manutenção não é necessariamente a mesma que em projetos de desenvolvimento. Entretanto é necessário que haja um acompanhamento da produtividade baseado em impacto, qualidade e complexidade. Esta escala deve ser configurada no início da fase de manutenção e deve ser monitorada constantemente visando assegurar-se que não está diminuindo significativamente. Uma produtividade decrescente está normalmente associada à complexidade crescente e qualidade decrescente;

- **Prazos de entrega**: Os prazos de entrega acordados devem ser mantidos e os atrasos não devem aumentar. O tamanho dos atrasos em termos de dias no calendário é uma métrica que pode ser usada para avaliar a pontualidade das operações de manutenção. Normalmente o intervalo entre as entregas é parte do acordo com o usuário e, uma vez estabelecido, deve ser mantido. Os desvios dos prazos estabelecidos costumam ser indícios de degradação do serviço. Isto torna a pontualidade do projeto um objetivo muito importante;
- **Satisfação do usuário**: A satisfação do usuário deve ser mantida pelo menos no mesmo nível, senão aumentada. Entretanto, medir o nível de satisfação do usuário não é uma tarefa simples. Esta tarefa está mais para as ciências sociais do que para a ciência da computação e exige investimento de tempo e recursos. Qualquer métrica de satisfação do usuário deve estar fundamentada na sensação do usuário sobre o que deveria ser oferecido em comparação com o que realmente é entregue. Esta medição deve ser realizada periodicamente. O decréscimo da satisfação do usuário também está relacionado a degradação do serviço;
- **Lucratividade:** A operação de manutenção deve ser lucrativa, ou pelo menos cobrir seus custos. Este objetivo deve ser analisado do ponto de vista contábil. A cobrança pelo suporte e por correções do sistema mais a receita extra das alterações e melhorias somadas devem pelo menos cobrir os custos das operações de manutenção. Os usuários devem ser alertados para o fato de que até mesmo as menores alterações em um software existente implicam em custos, tanto diretos como indiretos. Portanto este objetivo é tanto um desafio contratual, quanto um desafio para quem realiza as estimativas para as tarefas de manutenção.

#### *2.2.5.2 Modelo para Avaliação de Sistemas Legados*

A tomada de decisão sobre como evoluir e manter um sistema legado não pode ser feita de forma espontânea, pois exige justificativa válida que deve ser baseada em diversos fatores, incluindo o valor do software, análise de risco e estimativas de custo. Para dar subsídios a um processo de tomada de decisão em manutenção de software, [LUC01] propõe um modelo para avaliação de sistemas legados.

Este modelo apresenta um guia de referência para a captura de informações relevantes sobre o estado do sistema legado visando proporcionar uma tomada de decisão bem sucedida. O modelo hierárquico proposto representa tanto o valor técnico do produto quanto seu valor de negócio. O valor de negócio depende de quão eficiente e efetivo é o uso do sistema na organização e qual o benefício que traz para seus usuários. Já o valor técnico depende das características dos componentes e do ambiente técnico externo ao sistema, incluindo hardware e a infra-estrutura de suporte da organização.

Os atributos considerados como representativos do valor de um sistema legado são listados na Tabela 2. Para medir cada atributo, alguma forma de representação concreta deve ser escolhida. Estas representações, chamadas aqui de variáveis, também são listadas na tabela.

Tabela 2 – Modelo para Avaliação de Sistemas Legados [LUC01]

| **Atributo** | **Variável** | **Métrica** |
|---|---|---|
| Valor de Negócio | Valor Econômico | Valor de mercado, Índice de lucratividade, Taxa de retorno interna. |
| | Valor dos Dados | Percentual de arquivos de missão crítica, Percentual de arquivos dependentes da aplicação. |
| | Utilidade | Taxa de cobertura das funções de negócio, freqüência de uso, métricas de satisfação de cliente. |
| | Especialização | Percentual de funções altamente especializadas, Percentual de funções genéricas. |
| Valor Técnico | Facilidade de Manutenção | Lines of Code (LOC), Function Points (FP), Nós de controle de fluxo, complexidade, taxa de código 'morto'. |
| | Facilidade de Decomposição | Modularidade da arquitetura, percentual de módulos com separação de conceitos. |
| | Deterioração | Aumento do *Backlog*, Aumento da taxa de falhas, Aumento do tempo de resposta, Aumento do tempo de manutenção por requisição. |
| | Obsolescência | Idade do sistema, versão de sistema operacional, versão de hardware, disponibilidade de suporte técnico. |

Como valor de negócio, as variáveis de valor econômico, valor dos dados, utilidade e especialização devem ser consideradas. O valor econômico foi incluído para levar em conta a dimensão econômica do valor do software, ou seja, o quanto ele contribui para melhorar os lucros e diminuir os custos. O valor dos dados representa a relevância da informação gerenciada pelo sistema. Se o dado é essencial para o sucesso do negócio, então seu valor será considerado crítico. O valor de utilidade do software contabiliza o quão usado é o mesmo e como ele fornece suporte na busca pelas metas de negócio. Finalmente, a variável de especialização representa o grau de especialização do software em relação ao domínio de negócio e expressa quanto ele pode ou não ser substituído por equivalentes comerciais.

Como valor técnico, a facilidade de manutenção, facilidade de decomposição, deterioração e obsolescência devem ser consideradas. A facilidade de manutenção é considerada por fornecer

uma métrica agregada de vários atributos do software, tais como sua complexidade, facilidade de teste e facilidade de ser analisado. A facilidade de decomposição tenta capturar a qualidade da arquitetura do software considerando se, e em que extensão, os módulos do sistema implementam a separação de conceitos, tal como apresentada na seção 2.2.2.2. A obsolescência expressa o envelhecimento do sistema e a dificuldade para adaptá-lo às mudanças. Finalmente, a deterioração representa o envelhecimento do sistema resultante das contínuas alterações as quais foi submetido.

Para medir cada uma das variáveis o método define que deve ser estabelecida pela organização uma definição operacional na forma de uma seqüência de passos necessários para a medição. As métricas sugeridas para cada variável também são apresentadas na Tabela 2. As técnicas para avaliação incluem entrevistas com a gerência, análise dos custos de manutenção e técnicas de estimativa de custos.

## 2.3 Engenharia de Requisitos

De acordo com [KOT98] e [SOM97], ER é um termo que engloba todas as atividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos para um sistema computacional. O uso do termo "engenharia" implica que técnicas sistemáticas e repetíveis devem ser utilizadas para garantir que os requisitos do sistema sejam completos, consistentes e relevantes.

Nesta seção a ER será, inicialmente, contextualizada na engenharia de sistemas. Após, serão apresentados os principais conceitos relacionados à ER. A seguir serão descritas as principais etapas do processo de ER segundo [KOT98] e será apresentada a gerência de requisitos. A seguir serão apresentados os principais desafios da ER em ambientes de DDS. Por fim, serão apresentadas as conclusões quanto a ER.

### 2.3.1 Contextualização na Engenharia de Sistemas

De acordo com [KOT98], para muitos sistemas computacionais é impossível separar os requisitos do software dos requisitos do sistema como um todo. Além do software, um sistema pode incluir soluções de hardware e definições de processos a serem usados em conjunto quando o sistema estiver implantado, de forma que, segundo [BER04], apesar de o foco da criação de um sistema computacional ser a criação do código, não se pode esquecer do sistema como um todo.

Um sistema computacional pode pertencer, do ponto de vista da ER, a uma das seguintes categorias [LEF03][KOT98]:

- Sistemas computacionais desenvolvidos e vendidos como produtos comerciais.

Nesta categoria normalmente não há uma especificação explícita de requisitos. Os requisitos são criados pelas organizações que desenvolvem estes sistemas com base em suas percepções do mercado e da necessidade dos potenciais consumidores;

- Sistemas computacionais personalizados desenvolvidos com base em um conjunto de requisitos estabelecidos por uma organização. Esta categoria inclui o desenvolvimento de Sistemas de Informação, Sistemas Embarcados e Sistemas de Comando e Controle.

Além disto, em [INC04] a Engenharia de Sistemas é definida como uma abordagem interdisciplinar que visa capacitar a realização bem sucedida de sistemas, tendo seu foco na definição logo no início do ciclo de desenvolvimento das necessidades do cliente e das funcionalidades necessárias, documentando os requisitos e prosseguindo com o projeto e validação do sistema considerando o problema como um todo. Ainda de acordo com esta definição a Engenharia de Sistemas considera tanto as necessidades de negócio quanto as necessidades técnicas de todos os clientes, com o objetivo de fornecer um produto de qualidade.

Portanto, dado que a ER lida com a identificação, a compreensão e a documentação das necessidades do cliente, sejam elas no escopo do sistema ou do software, torna-se desnecessário diferenciar sua aplicação no desenvolvimento do sistema da sua aplicação no desenvolvimento do software. Desta forma, neste trabalho os termos "sistema" e "software" serão usados indistintamente, no sentido de sistema computacional baseado em software. Além disto, o foco deste trabalho será a segunda categoria de sistemas, ou seja, sistemas computacionais personalizados que necessitam da elaboração de um conjunto de requisitos para o seu desenvolvimento.

### 2.3.2 Conceitos

A ER apresenta uma diversidade de termos e conceitos. Alguns destes conceitos recebem definições diferentes e as vezes até contraditórias. Esta seção visa apresentar um conjunto básico de definições, necessárias à compreensão do assunto, e esclarecer quais serão adotadas neste trabalho quando houver divergência entre as definições para um mesmo termo.

#### *2.3.2.1 Requisitos*

Diversas definições para requisitos são encontradas na literatura. Neste trabalho será adotada a definição apresentadas por [THA90], onde requisitos são características do software necessárias para o usuário solucionar um problema de forma a atingir um objetivo ou ainda uma característica do software que deve ser realizada ou possuída por um sistema ou componente de sistema para satisfazer um contrato, padrão, especificação ou outra documentação formalmente imposta.

Entretanto, [GOG96] acrescenta mais um elemento a esta definição, que tem se mostrado importante na utilização de requisitos. Em sua visão, requisitos são propriedades que um software deve possuir para funcionar com sucesso no ambiente onde será utilizado. Neste caso, o autor considera tanto o contexto social como o técnico do ambiente em que o software será utilizado. Esta contextualização é necessária, pois muitas das informações para os requisitos estão ligadas ao ambiente social dos usuários e gerentes, e podem ser informais ou tácitas.

Outro aspecto importante diz respeito aos tipos de requisitos. Diversos autores dividem os requisitos em dois tipos: funcionais (o que o software deve fazer) e não-funcionais (como o software se comporta em relação a alguns atributos observáveis, como performance, segurança, confiabilidade, etc.) [THA90][NUS00][FRA98][SID96][KRU00]. De acordo com [KRU00], requisitos funcionais são utilizados para expressar o comportamento de um software através da especificação das condições de entrada e saída que deve possuir. Já os requisitos não-funcionais são atributos desejados de qualidade que não são descritos pelos requisitos funcionais. Por fim, [THA90] acrescenta que os requisitos funcionais capturam a natureza da interação entre o componente e seu ambiente. Enquanto que os requisitos não-funcionais restringem os tipos de solução que podem ser consideradas.

#### *2.3.2.2 Engenharia de Requisitos*

Em [THA90] a ER é definida como a ciência e a disciplina preocupada com a análise e documentação dos requisitos, incluindo análise das necessidades e análise e especificação dos requisitos. Além disso, a ER fornece mecanismos apropriados para facilitar as atividades de análise, documentação e verificação.

Entretanto, existem críticas quanto ao uso do termo "Engenharia de Requisitos", pois alguns autores não consideram esta uma verdadeira "engenharia" [NUS00]. Até 1995, Pressman utilizava o termo "análise de requisitos", para referir-se ao processo como um todo [PRE95]. A partir de 2001 passou a utilizar o termo "Engenharia de Requisitos", considerando a análise de requisitos como uma de suas etapas [PRE01]. Outros autores, como [LEF03], continuam usando o termo "gerência de requisitos".

#### *2.3.2.3 Gerência de requisitos*

Atenção especial deve ser dada ao termo "gerência de requisitos". Alguns autores, como [LEF03], utilizavam ou ainda utilizam este termo como sinônimo para ER. Contemplando as atividades de elicitação, análise, documentação e validação dos requisitos, além do processo de gerenciar as alterações nos requisitos do software.

Entretanto, o mesmo termo também é utilizado por outros autores como [KOT98], [PRE01] e [SOM97] referindo-se apenas às atividades de gerenciamento das alterações nos requisitos. Neste caso, a gerência de requisitos é considerada como sendo apenas um subconjunto das atividades contempladas pela ER. Neste trabalho será utilizada esta última

definição.

#### *2.3.2.4 Stakeholder*

As diversas definições encontradas na literatura demonstraram-se convergentes quanto ao significado do termo *stakeholder*. Em [LEF03], *stakeholder* é definido como qualquer pessoa que possa ser afetada pela implementação de um sistema ou aplicação. Já para [KOT98] e [SOM97] são pessoas que serão afetadas pelo sistema e têm influência direta ou indireta sobre os requisitos do software. E, de acordo com [KRU00], o termo *stakeholder* é definido como qualquer pessoa ou representante de uma organização que possua um *stake* – um grande interesse – no resultado de um projeto.

Segundo [KRU00], o *stakeholder* mais óbvio é o usuário final, mas devemos lembrar de considerar outros *stakeholders* importantes: compradores, contratantes, desenvolvedores, gerentes de projeto ou quaisquer outros que se importem ou cujas necessidades devam ser satisfeitas pelo projeto. Adicionalmente, [SOM97] sugere que sejam considerados quaisquer indivíduos envolvidos nos processos organizacionais que possam ser afetados pelo software, tais como consumidores ou entidades externas como autoridades certificadoras, autoridades reguladoras e órgãos governamentais.

#### *2.3.2.5 Documento de requisitos*

De acordo com [SOM97], o documento de requisitos é uma declaração formal dos requisitos para os clientes, usuários finais e equipe de desenvolvimento do software. De acordo com [KOT98], para garantir que todas as informações essenciais serão contempladas, as organizações devem definir seus próprios padrões para o documento de requisitos estabelecendo que informações devam ser incluídas e em que formato.

1 Introdução
- 1.1 Propósito
- 1.2 Escopo
- 1.3 Definições, acrônimos e abreviações
- 1.4 Referências

2 Visão geral
- 2.1 Descrição geral
- 2.2 Perspectivas para o produto
- 2.3 Funções do produto
- 2.4 Características dos usuários
- 2.5 Restrições
- 2.6 Suposições e dependências

3 Requisitos específicos
4 Apêndices
5 Índice

Figura 11 - Estrutura sugerida por [IEE98]

Algumas organizações como o Departamento de Defesa dos Estados Unidos da América (EUA) e o *Institute of Electrical and Electronic Engineers* (IEEE) definiram seus próprios padrões para o documento de requisitos. Provavelmente, o mais acessível destes padrões seja o IEEE Std 830-1998 [IEE98]. Este padrão sugere a estrutura apresentada na Figura 11 para o documento de

requisitos.

### 2.3.3 Processo de Engenharia de Requisitos

Um processo de ER é um conjunto estruturado de atividades a serem seguidas para criar, validar e manter um documento de requisitos. Uma descrição completa de processo deve incluir quais atividades devem ser realizadas, a organização entre elas, quem é responsável por cada atividade, as entradas e saídas de cada uma e as ferramentas usadas para suportar a ER [SOM97].

Em [SOM97] é sugerido que cada organização deve desenvolver um processo adequado à sua realidade. Uma descrição adequada de processo fornece diretrizes às pessoas envolvidas e reduz a probabilidade de que atividades sejam esquecidas.

A Figura 12 apresenta o modelo de processo de ER proposto por [KOT98].

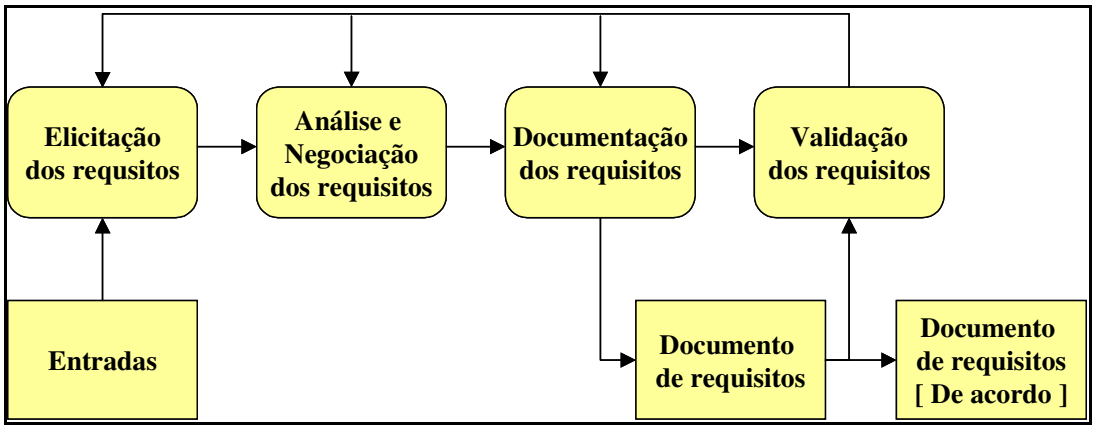


Figura 12 - Modelo de processo de ER (adaptado de [KOT98])

Apesar do aparente fluxo entre as atividades, não existe uma fronteira explícita entre estas atividades. Na prática existe muita sobreposição e interação entre uma atividade e outra. Além disto, em paralelo com este processo existe um outro processo de gerência de requisitos, concebido para permitir gerenciar as mudanças nos requisitos. A gerência de requisitos visa permitir a acomodação destas mudanças e garantir que estas mudanças sejam realizadas no documento de requisitos de uma forma controlada.

#### *2.3.3.1 Elicitação dos requisitos*

Elicitação de requisitos é o nome normalmente dado às tarefas envolvidas na descoberta dos requisitos do sistema. Os desenvolvedores e engenheiros do sistema trabalham em conjunto com os clientes e usuários finais para encontrar o problema a ser resolvido, os serviços exigidos do sistema, a performance desejada e as restrições de hardware. Isto não envolve apenas perguntar às pessoas o que elas desejam. Isto necessita uma cuidadosa análise da organização, do domínio de aplicação e dos processos de negócio nos quais o sistema será utilizado [KOT98].

As informações obtidas durante a elicitação dos requisitos serão interpretadas, analisadas,

modeladas e validadas para formarem um completo conjunto de requisitos do software em questão. Por isso, a elicitação de requisitos tem forte relação com as demais atividades da ER [NUS00].

Esta atividade consiste de uma tarefa complexa envolvendo os *stakeholders* do sistema. De acordo com [KOT98], alguns autores preferem o termo "descoberta de requisitos", pois este termo reflete melhor as dificuldades envolvidas.

Um dos mais importantes objetivos da elicitação dos requisitos é a definição do escopo do software. O escopo determina, de forma simples, como o software vai se integrar ao ambiente, sendo que esta sua definição afeta todo o trabalho de elicitação [NUS00]. Em conjunto com o escopo, são definidos os objetivos do software. O foco é direcionado para os objetivos e necessidades dos *stakeholders*, preocupando-se com o que o software deve realizar, sem preocupar-se em encontrar soluções antes de obter o domínio do problema.

#### *2.3.3.2 Análise e negociação dos requisitos*

O objetivo desta atividade é estabelecer um acordo entre *stakeholders* sobre um conjunto de requisitos que sejam completos, consistentes e livres de ambigüidade, uma vez que servirão como base para o desenvolvimento do sistema. Durante a atividade de análise são descobertos requisitos esquecidos, conflitantes, ambíguos, sobrepostos e irreais. Se estes problemas existirem, os *stakeholders* devem negociar e chegar a um acordo quanto à modificação dos requisitos.

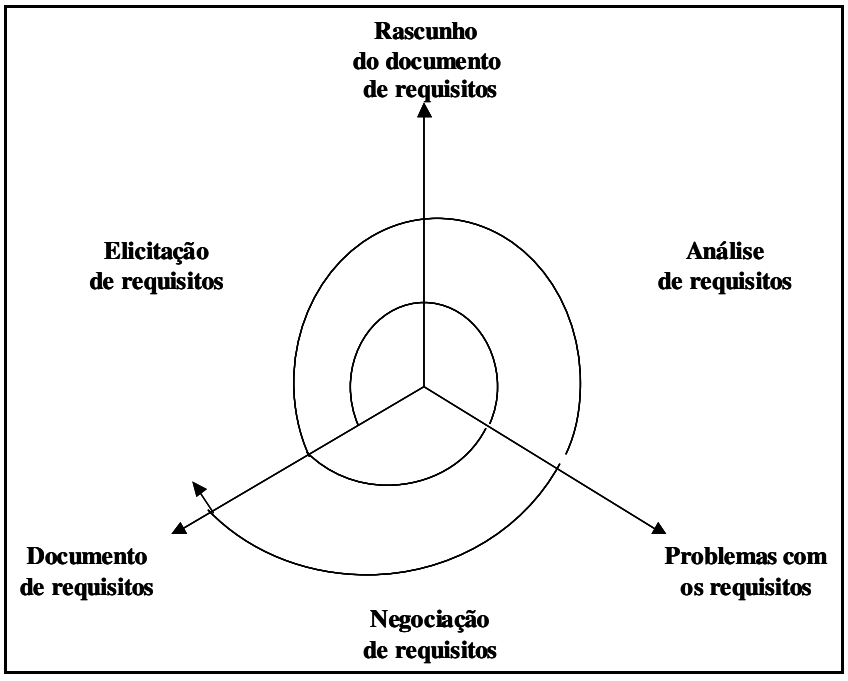


Figura 13 - Elicitação, análise e negociação (adaptado de [KOT98])

A análise e a negociação de requisitos são atividades fortemente ligadas à atividade de elicitação de requisitos. À medida que os requisitos são descobertos, durante a atividade de elicitação, alguma análise é inevitavelmente realizada. Assim, problemas podem ser reconhecidos,

discutidos com a fonte do requisito e resolvidos imediatamente. Apesar de serem classificadas neste modelo como atividades separadas, deve-se ter em mente que na realidade são atividades intercaladas. Portanto, as atividades de elicitação, análise e negociação podem ser encaradas como um processo em espiral, como demonstra a Figura 13.

#### *2.3.3.3 Documentação dos requisitos*

Os requisitos para os quais foi estabelecido um acordo entre *stakeholders* de que são completos, consistentes e livres de ambigüidade devem ser documentados em um nível adequado de detalhes.

Em geral é necessário que o documento de requisitos seja compreensível por todos os *stakeholders*. Para isto podem ser usados diagramas ou linguagem natural. Detalhes sobre o documento de requisitos podem ser encontrados na seção 2.3.2.5.

#### *2.3.3.4 Validação dos requisitos*

A validação de requisitos é a última etapa do processo de ER. O objetivo é validar os requisitos para garantir que eles representam uma especificação aceitável do que deve ser implementado. Esta atividade exige que *stakeholders* do sistema analisem os requisitos para assegurar que todos os requisitos foram definidos sem ambigüidades, inconsistências ou omissões, e que todos os erros foram detectados e corrigidos.

O principal mecanismo de validação de requisitos é a utilização de revisões técnicas formais. A equipe de revisão deve ser composta de engenheiros de software, usuários, clientes, e outros *stakeholders* que examinam a especificação do software buscando erros no conteúdo dos requisitos, possíveis interpretações errôneas, necessidade de esclarecimentos, informações omitidas, inconsistências, conflitos entre requisitos ou requisitos impossíveis de serem atingidos no ambiente em questão [PRE01].

Em diversos casos, é recomendada a realização de casos de teste. Idealmente, os requisitos devem ser testáveis. Nesta abordagem, testes para os requisitos são criados como parte do processo de validação. Se um teste é difícil ou impossível de ser projetado, em geral significa que os requisitos serão de difícil implementação e devem ser reconsiderados [SOM03].

As etapas de validação e análise têm grande importância dado que a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao re-trabalho, quando estes erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação.

### 2.3.4 Gerência dos requisitos

Este é o processo responsável pela gerência das mudanças nos requisitos do sistema. De acordo com [KOT98], as principais responsabilidades da gerência de requisitos são:

- Gerenciar mudanças em requisitos já aprovados;
- Gerenciar os relacionamentos entre os requisitos;
- Gerenciar as dependências entre o documento de requisitos e outros documentos produzidos durante o processo de engenharia do sistema.

Os requisitos evoluem devido a mudanças no ambiente do sistema e devido à evolução da compreensão dos *stakeholders* de suas reais necessidades. Novos requisitos surgem e requisitos existentes mudam em qualquer estágio do processo de desenvolvimento do sistema. Isto pode causar sérios problemas para os desenvolvedores do sistema. Para minimizar estas dificuldades é necessário controlar e documentar estas mudanças. O impacto das mudanças deve ser avaliado e, se as mudanças foram aceitas, o projeto e a implementação do sistema também devem ser modificados.

De forma geral, a gerência de requisitos envolve a utilização de técnicas e ferramentas para gerenciamento de configuração e controle de versão, além de verificar inconsistências nas especificações, conforme estas evoluem [NUS00]. Algumas mudanças podem afetar profundamente o tempo e o custo de um projeto, se suas definições apontarem características fundamentais do software [REI00].

Além disto, de acordo com [KOT98], os requisitos não podem ser efetivamente gerenciados sem rastreamento (*Traceability*). Diz-se que um requisito pode ser rastreado se é possível determinar quem sugeriu o requisito, porque o requisito existe, a quais outros requisitos ele está relacionado e como ele está relacionado com outras informações como artefatos de projeto, implementação e documentação de usuário. O rastreamento também permite encontrar outros requisitos que podem ser afetados quando uma mudança é solicitada.

#### *2.3.4.1 Identificação e armazenamento dos requisitos*

Um pré-requisito essencial para a gerência de requisitos é de que todos os requisitos devem possuir algum tipo de identificador único. Segundo [KOT98], a ausência de identificação única dos requisitos pode tornar a gerência de requisitos impraticável.

Uma prática muito comum é numerar os requisitos de acordo com os capítulos e seções do documento de requisitos onde ele se encontra. Existem dois problemas com este tipo de identificação:

- É impossível atribuir um número único até que o documento esteja concluído, pois os capítulos e seções precisam estar estáveis. Quando um requisito é descoberto, pode não estar claro ainda onde ele deve ser encaixado no documento de requisitos. Portanto, ele não pode receber um número e não pode ser referenciado pelos demais requisitos;
- Associar um identificador baseado em número de capítulo e seção classifica implicitamente o requisito. Isto pode sugerir que o requisito está fortemente

relacionado com outros requisitos com um identificador similar, ou seja, do mesmo capítulo e seção. Os leitores do documento podem acabar pensando que não existem outros relacionamentos importantes com requisitos de outras partes do documento.

Existem algumas alternativas para resolver este problema:

- **Re-numeração automática:** muitos softwares de processamento de texto permitem a numeração automática de parágrafos e a inclusão de referências cruzadas. Assim pode-se atribuir um número ao requisito a qualquer momento. À medida que os requisitos são reorganizados o software vai atualizando a numeração bem como todas as referências ao requisito nos demais requisitos presentes no documento;
- **Identificação com base em registro de banco de dados:** quando um requisito é identificado ele é imediatamente inserido em um banco de dados e um identificador de registro é atribuído. O identificador de registro será usado para referenciar o requisito;
- **Identificação simbólica:** o requisito pode ser identificado a partir de um nome simbólico associado ao requisito em si. Por exemplo, FR1 e FR2 podem ser requisitos funcionais enquanto que NF3 e NF4 podem ser requisitos não funcionais. O problema desta abordagem é que muitas vezes pode ser difícil classificar um requisito e atribuir um mnemônico útil a ele.

As vantagens de armazenar os requisitos em documentos de requisitos é que os requisitos estão reunidos em um único documento. Isto torna fácil o acesso aos mesmos e a confecção de novas versões do documento de requisitos. Entretanto, de uma perspectiva de ER, existem várias desvantagens nesta abordagem:

- Informações sobre dependências de requisitos (informações de rastreamento) são mais difíceis de serem mantidas;
- Os recursos de pesquisa são limitados às funcionalidades de busca textual oferecidas pelos softwares de processamento de textos. Não é fácil procurar, por exemplo, por grupos de requisitos com características semelhantes;
- Não é fácil ligar eletronicamente os requisitos com as modificações propostas;
- Qualquer controle de versão só é possível no nível do documento de requisitos como um todo, ou pelo menos, de capítulos individuais. É impossível manter diferentes versões do mesmo requisito;
- Não é possível navegar automaticamente entre requisitos relacionados.

Para oferecer estes recursos os requisitos precisam ser armazenados em um banco de dados, com cada requisito representado por uma ou mais entidades de banco de dados. Os

recursos oferecidos por sistemas de gerenciamento de banco de dados (SGBD) podem ser utilizados para ligar requisitos relacionados e geralmente é possível criar consultas complexas para recuperar grupos de requisitos. O SGBD pode oferecer também recursos de controle de versão, ou pelo menos permitir que estes recursos sejam implementados. Além disto, SGBD's normalmente possuem ferramentas de geração de relatórios que podem ser usados para desenvolver geradores de rascunhos de documentos de requisitos. Os sistemas comerciais de gerência de requisitos normalmente fazem uso de algum SGBD comercial para o armazenamento dos requisitos.

#### *2.3.4.2 Gerência de mudança dos requisitos*

A gerência de mudanças envolve procedimentos, processos e padrões a serem usados para gerenciar as mudanças nos requisitos do sistema. Deve garantir que informações similares sejam coletadas para cada proposta de mudança de requisitos e que o julgamento seja feito com base em uma análise de custo e benefício de cada proposta. Sem uma gerência de mudanças formal não é possível garantir que as mudanças propostas suportam os objetivos de negócio fundamentais do sistema.

Para garantir uma abordagem consistente a organização deve estabelecer um conjunto de políticas que englobe as seguintes definições:

- O processo de requisição de mudanças (*Change Requests*) e as informações necessárias para processar cada requisição;
- O processo usado para a análise de impacto e custo-benefício da mudança tanto dos requisitos propostos quanto das informações relacionadas no rastreamento;
- Os membros do grupo que trata formalmente das requisições de mudança. É importante manter um grupo tão independente quanto possível para que sejam objetivas as decisões sobre a contribuição das mudanças propostas para os objetivos gerais do sistema;
- O tipo de suporte de software necessário para o processo de controle de mudanças.

O processo de gerência de mudanças consiste de um conjunto de atividades que visam documentar, registrar, analisar, orçar e implementar as mudanças propostas para um conjunto de requisitos. Consiste basicamente das três etapas demonstradas na Figura 14.

**Análise do problema e especificação das modificações**: algum problema com os requisitos é identificado. Isto pode decorrer de uma análise dos requisitos, nova necessidade dos *stakeholders* ou de um problema operacional com o sistema. Os requisitos devem ser analisados usando-se as informações sobre o problema e uma mudança de requisitos deve ser proposta.

**Análise e orçamento das modificações**: a proposta de modificação é analisada para verificar quantos requisitos devem ser modificados e quanto isto irá custar.

**Implementação das modificações**: a modificação é implementada. É realizado um conjunto de correções no documento de requisitos ou uma nova versão do documento é produzida.

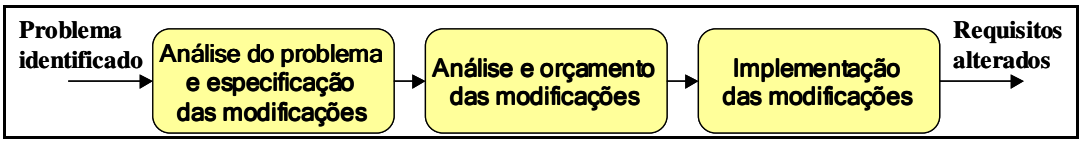


Figura 14 - Etapas da gerência de mudança dos requisitos (adaptado de [KOT98])

Os processos de análise do problema e implementação das modificações são dependentes do tipo de modificação, dos requisitos afetados e do tipo de documento sendo utilizado. Entretanto, a análise e o orçamento das modificações são descritos por [KOT98] como um processo mais genérico, tal como apresentado na Figura 15.

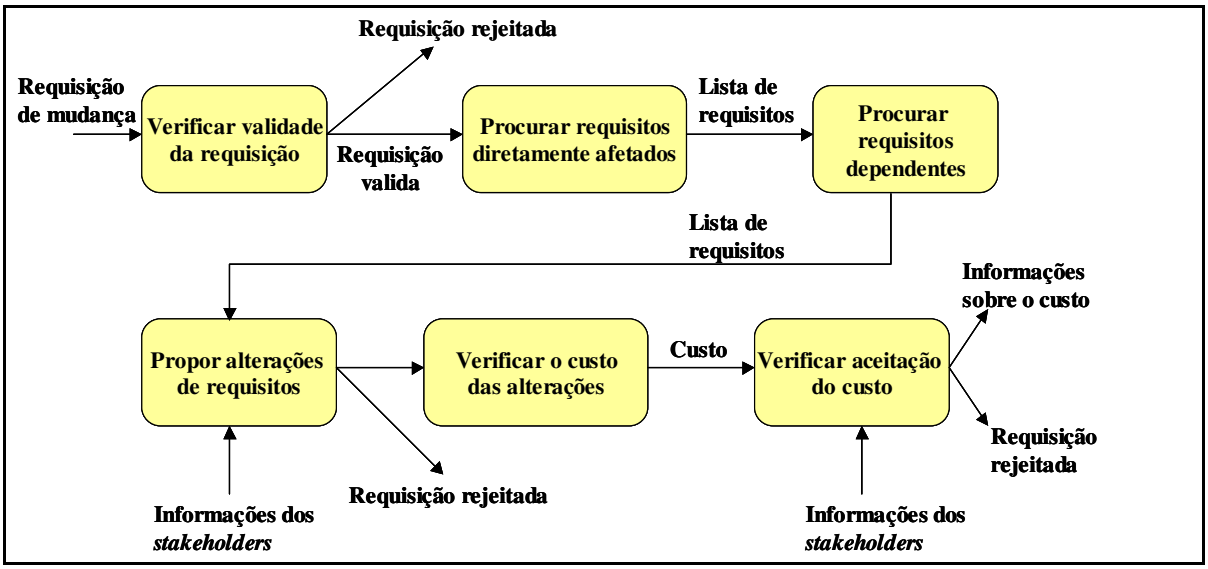


Figura 15 - Análise e orçamento das modificações (adaptado de [KOT98])

Segundo [KOT98], existem seis atividades neste processo:

- A requisição de mudança é verificada quanto a sua validade, pois algumas vezes os *stakeholders* têm uma compreensão incorreta dos requisitos e propõem mudanças desnecessárias;
- Os requisitos que são diretamente afetados pela mudança são descobertos;
- As informações de rastreamento são utilizadas para encontrar requisitos dependentes que também são afetados pela mudança proposta;
- As modificações reais que devem ser feitas são propostas. Os *stakeholders* devem ser consultados nesta etapa para verificar se concordam com estas modificações;
- Os custos envolvidos na mudança são estimados;
- Negociações com os *stakeholders* são conduzidas para verificar se os custos envolvidos são aceitáveis.

#### *2.3.4.3 Rastreamento de requisitos*

Um aspecto crítico do processo de gerência dos requisitos é a avaliação do impacto que a mudança acarretará no resto do sistema. Se uma mudança é proposta enquanto os requisitos ainda estão sendo elaborados é possível determinar como os demais requisitos serão afetados. Se a mudança é proposta quando o sistema já está sendo implementado será necessário determinar como a mudança afeta os requisitos, o projeto do sistema e sua implementação. Caso a proposta seja feita quando o sistema já está implantado será necessária uma checagem adicional: verificar como cada um dos *stakeholders* será afetado pela implementação da mudança.

Para permitir este tipo de análise de impacto é necessário manter informações suplementares sobre as dependências dos requisitos e a implementação dos mesmos. Estas informações são normalmente chamadas de informações de rastreamento (*traceability*). A análise de impacto depende destas informações para descobrir quais requisitos serão afetados pela modificação proposta. Em [DAV93] é sugerida a classificação em quatro tipos de informações de rastreamento (Figura 16):

- **Para trás – de (*Backward-from*):** referência do requisito para suas origens em outros documentos ou pessoas;
- **Para frente – de (*Forward-from*):** referência do requisito para os artefatos de projeto e implementação;
- **Para trás – para (*Backward-to*):** referência dos artefatos de projeto e implementação de volta para os requisitos;
- **Para frente – para (*Forward-to*):** referência de outros documentos, que precedem os requisitos, para os requisitos do sistema.

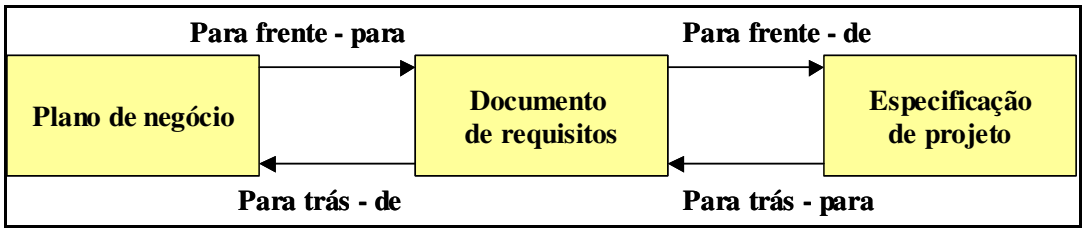


Figura 16 - Tipos de rastreamento (adaptado de [KOT98])

Esta classificação pode gerar um volume grande de informações. Na prática, segundo [KOT98], pode ser muito caro coletar e gerenciar todo tipo de informação de rastreamento, tornando necessário definir-se políticas de rastreamento que estabeleçam quais informações são essenciais e devem ser mantidas.

Adicionalmente, [KOT98] considera muito importante a manutenção do rastreamento das dependências entre os próprios requisitos. Estas informações podem ser consideradas como extensões aos tipos de rastreamento "Para trás – de" e "Para frente – para" de modo a permitir referências ao próprio documento de requisitos. Outra maneira de manter o rastreamento das

dependências entre os requisitos é o uso de tabela de rastreamento (*Traceability Table*). Em uma tabela de rastreamento os requisitos são listados ao longo dos eixos X e Y e as dependências entre eles são marcadas nas células da tabela, tal como demonstrado na Figura 17.

| para: / de: | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| R1 | | X | X | |
| R2 | | | | X |
| R3 | X | | | |
| R4 | | | X | |

Figura 17 - Exemplo de tabela de rastreamento (adaptado de [KOT98])

Neste exemplo os identificadores dos requisitos são usados como títulos das colunas e linhas e uma marca 'X' é usada nas células que representam uma relação entre dois requisitos. Também é possível especificar a hierarquia da dependência, ou seja, explicitar se um requisito X depende ou tem como dependente de si um requisito Y. Para isto basta convencionar que os requisitos nas linhas, por exemplo, tem como dependentes os requisitos das colunas marcadas com 'X'. Desta forma, no exemplo da Figura 17, pode-se ler que o requisito R1 tem como dependentes os requisitos R2 e R3 e que o requisito R3 depende dos requisitos R1 e R4. Portanto, se uma proposta de modificação do requisito R1 for recebida, o impacto da modificação sobre os requisitos R2 e R3 também deverá ser analisado.

## 2.4 Desenvolvimento Distribuído de Software

Observou-se na última década um grande investimento na conversão de mercados nacionais em mercados globais, criando novas formas de competição e colaboração entre os países [CAR99]. Entretanto, o mercado global de software vinha passando por diversas crises. Por um lado, um grande número de falhas em projetos. De outro, uma crescente demanda, atingida pela escassez de recursos capacitados. Nesse contexto, muitas organizações encontraram no DDS uma alternativa, experimentando o desenvolvimento de software com equipes geograficamente distantes entre si. Atualmente um grande número de organizações desenvolve software com equipes distribuídas [KAR98], [KIE03], [PRI04].

Entre os fatores que têm contribuído para o crescimento do DDS, podemos destacar o custo mais baixo e disponibilidade de mão de obra; a evolução e maior acessibilidade a recursos de telecomunicação; a evolução das ferramentas de desenvolvimento; a necessidade de possuir recursos globais para utilizar a qualquer hora; as vantagens de estar perto do mercado local; a formação de equipes virtuais para explorar as oportunidades de mercado; e a pressão para o desenvolvimento *time-to-market*, utilizando as vantagens proporcionadas pelo fuso horário diferente, no desenvolvimento conhecido como *round-the-clock*, ou seja, o desenvolvimento quase

que contínuo.

Neste sentido, o DDS tem atraído um grande número de pesquisas na área de Engenharia de Software [PRI04], [EVA00], [EVA05], [LOP03], [LOP03b], [LOP04], [ESP05] e [ESP05b]. Os engenheiros de software têm reconhecido a grande influência desta nova forma de trabalho no seu dia-a-dia e estão em busca de modelos que facilitem o desenvolvimento de software com equipes geograficamente distantes. São freqüentes os esforços que os pesquisadores têm feito no intuito de entender os fatores que permitem organizações multinacionais ou virtuais a obterem sucesso trabalhando através das fronteiras físicas e culturais dos países. Em suma, existe uma série de problemas e desafios inerentes ao desenvolvimento de software. E o DDS, ao acrescentar fatores como dispersão geográfica e dispersão temporal acentuou alguns dos desafios existentes e acrescentou novos desafios ao processo de desenvolvimento.

Alguns autores utilizam o termo desenvolvimento global de software (DGS) para referenciar o desenvolvimento de software onde as equipes estão distribuídas ao redor do mundo e o termo DDS para cenários onde as equipes estão distribuídas em pequenas distâncias, tais como prédios ou cidades distintas. Nesta dissertação de mestrado estes termos serão utilizados como sinônimos, sendo usado preferencialmente o termo DDS.

### 2.4.1 Engenharia de Requisitos em ambientes de DDS

A crescente globalização do ambiente de negócios tem afetado diretamente o mercado de desenvolvimento de software [HER01]. Em busca de vantagens competitivas como baixo custo, alta produtividade e qualidade na área de desenvolvimento de sistemas, diversas organizações optaram por distribuir o processo de desenvolvimento de software dentro de seu país, ou em outros países, como Índia, Irlanda e Brasil. Essas regiões oferecem, muitas vezes, incentivos fiscais ou possuem grande concentração de massa crítica em determinadas áreas.

Entretanto, os desafios apresentados pela distribuição da equipe envolvida no processo de software são significativos. Questões como diferenças culturais e de fuso horário devem ser analisadas, visando evitar impactos negativos.

Nesse contexto, de acordo com [LOP03], a ER também é influenciada pela distribuição das equipes. O processo de requisitos, mesmo em ambientes co-localizados, é apontado como crítico no desenvolvimento de software [SOM97]. Mas ao lidar com ambientes de DDS, as dificuldades com requisitos tendem a se exacerbar, levando alguns autores [DAM02][ZOW02] a sugerir a necessidade de um novo processo de requisitos para DDS.

Segundo [LOP03b], o DDS promove um aprofundamento das dificuldades inerentes ao processo de desenvolvimento de software, ou ainda, o surgimento de novas dificuldades, como diferenças culturais e linguagem. Como a distância envolvida pode compreender países distantes, comumente, a linguagem e a cultura são diferentes. Com isso, os problemas causados por

ambigüidade e falta de clareza nos requisitos são intensificados. A compreensão dos requisitos ao serem lidos em uma língua diferente da nativa é mais limitada, levando a interpretações incorretas. Diferenças culturais como atitude em relação à hierarquia, riscos e valores culturais podem ampliar a possibilidade de conflitos.

Sem o conhecimento presencial do ambiente onde o software será inserido, a compreensão das razões e expectativas do software por parte da equipe de desenvolvimento é reduzida [DAM02]. A comunicação também apresenta novos desafios. Com a perda de contato face a face entre a equipe de desenvolvimento e os usuários, existe uma maior dificuldade de esclarecimento em caso de dúvidas. Além disso, com a utilização de meios de baixo contexto, como correio eletrônico, o contato informal entre os membros dos diversos grupos é limitado, reduzindo a confiança entre eles.

Em casos de grupos separados por diversos fusos horários, em geral, ocorre uma maior demora na tomada de decisões. Uma simples troca de mensagens por correio eletrônico para esclarecimento de um requisito pode levar dias se os horários de trabalho não coincidirem.

Reuniões por vídeo ou teleconferência podem não ser tão efetivas. Grupos com reduzida confiança tendem a evitar comprometimentos. Algumas culturas valorizam mais questões como pontualidade e agenda que outras, ampliando as possibilidades de conflitos.

Em [LOP03b] é apresentada uma síntese das principais dificuldades identificadas na ER em ambientes de DDS, a saber:

- Identificação dos *stakeholders*;
- Ambigüidade e falta de clareza;
- Compreensão do ambiente onde os requisitos se aplicam;
- Comunicação;
- Confiança;
- Demora;
- Baixa efetividade nas reuniões por vídeo ou teleconferência;
- Conflitos;
- Diferenças culturais.

### 2.4.2 Desafios para ER em Manutenção de Software em DDS

O processo de ER exige a manipulação de uma grande quantidade de informações. Tipicamente, além dos documentos de requisitos propriamente ditos, é necessária também a manutenção de informações sobre controle de versão, relatórios de análise de impacto, estimativas de custos e todas as informações necessárias à gerência do processo de submissão e apreciação das requisições de mudança de requisitos. Fazendo com que estes processos tenham

uma grande capacidade de produzir e consumir conhecimento.

No contexto de manutenção de software, o conhecimento sobre requisitos é particularmente importante. Segundo Liu e seus colaboradores [LIU99], compreender os requisitos de sistemas legados torna-se importante para:

- Compreender os *stakeholders*, os usuários e o contexto de negócio do sistema;
- Obter uma visão geral das funcionalidades e do ambiente do sistema;
- Compreender as intenções por trás do projeto original do sistema;
- Obter conhecimento sobre a interação do sistema com seus usuários no suporte às operações de negócio;

Este conhecimento é útil para melhorar o sistema legado, integrá-lo com o restante dos sistemas de informação ou realizar a reengenharia do sistema. Infelizmente, nem sempre é possível obter este conhecimento. De acordo com diversos autores [EBN02][PRE01][WHI95][ZAN03][LIU99], a ausência de documentação e a indisponibilidade dos *stakeholders* originais é um cenário muito comum na prática de manutenção de sistemas legados. Muitas vezes, o sistema em si, seu código-fonte e seus usuários são as únicas fontes de conhecimento sobre tais sistemas.

Um outro fator negativo para gerência de requisitos em projetos de manutenção é que a ER é, geralmente, discutida como uma fase inicial no desenvolvimento de software [ZAN03]. Entretanto, o conhecimento dos requisitos requer um esforço contínuo no refinamento progressivo das exigências contidas nas regras de negócio das organizações e em atendimento às necessidades dos *stakeholders*, durante todo o ciclo de vida do software. Assim, os requisitos devem ser gerenciados de forma a evoluírem em sincronia com o software. Práticas de ER que vinculem os requisitos ao escopo de projetos, ao invés do produto de software, podem levar a uma fragmentação das especificações de requisitos e dificultar a evolução e a transferência do conhecimento sobre a aplicação legada.

Além disto, em ambientes de DDS fatores como diferenças culturais e de idioma dificultam o processo de transferência de conhecimento sobre a aplicação entre os diversos *stakeholders* geograficamente distribuídos. De acordo com Damian [DAM02] e Lopes [LOP03b], a gestão de conhecimento influencia a ER em ambientes de DDS. A ER em DDS envolve, por exemplo, documentos cujo conteúdo pode ter diversas origens. Captar, processar, armazenar e disponibilizar globalmente o conhecimento gerado e consumido pelos processos de manutenção e pela ER são questões que podem ser endereçadas pela gestão do conhecimento.

Cabe salientar também que os desafios aqui descritos não estão restritos ao escopo da manutenção de software, sendo muitas vezes causados por decisões tomadas e práticas adotadas durante o projeto de desenvolvimento de software. Mas é sob a perspectiva da manutenção de software que estes desafios são evidenciados. Uma parte significativa do conhecimento gerado

durante o desenvolvimento de um novo produto de software, por exemplo, não é reaproveitada durante o projeto inicial. Portanto, a decisão de não preservar este conhecimento não acarreta nenhum prejuízo neste momento, podendo inclusive ser considerada como uma forma de economia de recursos e, consequentemente, redução de custos. Mas a ausência deste mesmo conhecimento poderá acarretar dificuldades em futuros projetos de manutenção de software, principalmente se o projeto de manutenção for executado por outra equipe e em outra localidade.

# 3 Estudos Relacionados

Após a pesquisa bibliográfica realizada, um reduzido número de artigos sobre a ER em DDS foi encontrado. Este capítulo apresenta uma compilação dos principais estudos relacionados ao tema de ER em DDS.

## 3.1 Estudo de Prikladnicki, Audy e Evaristo

O artigo escrito por Prikladnicki, Audy e Evaristo [PRI03] teve como objetivo analisar que tipos de problemas os times de projetos enfrentam quando gerenciam requisitos em um ambiente distribuído de desenvolvimento de software e como estes problemas são tratados. Com este objetivo, um estudo de caso foi realizado em uma organização que mantém centros de desenvolvimento de software no Brasil, na Rússia e na Índia. O artigo apresenta resultados iniciais deste estudo de caso do processo de gerência de requisitos no desenvolvimento global de software (DGS), sob um contexto SW-CMM (*Software Capability Maturity Model*).

Neste estudo de caso, foram acompanhados dois projetos. Os projetos foram estudados sob o ponto de vista do centro de desenvolvimento localizado no Brasil. A organização utiliza como base para o processo de desenvolvimento de software o MSF e o SW-CMM.

O primeiro projeto tinha como objetivo desenvolver um novo sistema para a área de recursos humanos da organização e seria utilizado globalmente. Este projeto durou nove meses. No Brasil, os membros da equipe de projeto estavam localizados em diferentes prédios ocupados pelo centro de desenvolvimento de software estudado. Tanto os clientes quanto os usuários finais estavam localizados em prédios distintos da matriz da organização nos EUA.

Já o segundo projeto tinha como objetivo unificar e consolidar em uma única aplicação duas versões (utilizadas em diferentes países) de uma aplicação da área de manufatura. Este projeto durou um ano. O projeto teve a participação de trinta pessoas localizadas no mesmo prédio do centro de desenvolvimento de software no Brasil. Já os clientes estavam distribuídos por em países como Brasil, México, Argentina e Canadá. Sendo que os usuários finais encontravam-se na América Latina e no Canadá.

A implementação do nível 2 do SW-CMM no centro de desenvolvimento brasileiro, deu origem a algumas diferenças de processo com relação às equipes de outras localidades, que não estavam envolvidas na certificação. Isto ampliou o tempo de algumas atividades, devido à necessidade de explanações a respeito do processo.

Por outro lado, a implementação do SW-CMM colaborou em grande escala para minimizar os problemas encontrados neste cenário. A definição de um processo de desenvolvimento de software trouxe excelente resultados relacionados aos problemas de ambientes distribuídos de desenvolvimento de software. Os times de projeto também foram capazes de padronizar todo o trabalho e de convergir para um entendimento comum sobre a melhor abordagem para conduzir ambos os projetos. Assim, os autores concluirão que muitos dos esforços feitos no processo de certificação para o nível 2 do SW-CMM contribuíram para minimizar os problemas relacionados a organização, padronização e, algumas vezes, comunicação.

Como impactos do DGS no gerenciamento de requisitos, concluiu-se que essa pode ser uma tarefa árdua se os processos de software não estiverem bem definidos e se as equipes não estiverem preparadas para trabalhar neste cenário. O treinamento em aspectos não técnicos, tais como confiança e diferenças culturais, auxiliou na redução de problemas decorrentes da distância entre os times de projeto. As lições aprendidas com os estudos de caso estão descritas no Tabela 3.

Tabela 3 – Lições aprendidas em DGS em SW-CMM [PRI03].

| **Nº.** | **Lição** |
|---|---|
| 1 | Treinar a equipe em assuntos como confiança, cultura, comunicação e colaboração, por exemplo, é essencial; |
| 2 | A padronização do trabalho é obrigatória; |
| 3 | Encontros freqüentes com pessoas geograficamente distantes são importantes para rastrear o projeto; |
| 4 | Um processo bem definido é a chave para o sucesso; |
| 5 | Se possível, viagens devem ser feitas para que as equipes envolvidas se conheçam; |
| 6 | A diferença de fuso horário pode ser, ao mesmo tempo, uma vantagem e uma desvantagem; |
| 7 | O uso de ferramentas como correio eletrônico, teleconferência e videoconferência é muito importante; |
| 8 | Um processo de certificação como SW-CMM nível 2 pode ampliar o overhead; |
| 9 | O modelo SW-CMM nível 2 auxiliou a definir uma forma padrão de trabalho; |
| 10 | É muito importante conhecer as pessoas com quem se está trabalhando, considerando as formas de comunicar, as diferenças culturais, etc. |

## 3.2 Estudos de Damian e Zowghi sobre ER em DDS

Os estudos de Damian e Zowghi têm como foco principal o entendimento e descrição do impacto exercido pela distância na negociação de requisitos de software. Em [DAM02], é apresentada a relação entre cultura, conflitos e distância na negociação de requisitos de forma distribuída globalmente. Para isso foi conduzida uma pesquisa baseada em estudos de caso, visando esclarecer o impacto causado pela distribuição dos *stakeholders* nas atividades de ER em ambientes de DGS.

Como resultado foi construído um modelo dos desafios apresentados pela distribuição dos stakeholders à ER, conforme apresentado na Figura 18. A camada superior do modelo apresenta os quatro maiores problemas da distribuição geográfica dos *stakeholders*: comunicação inadequada, gerência do conhecimento, diversidade cultural e diferença temporal.

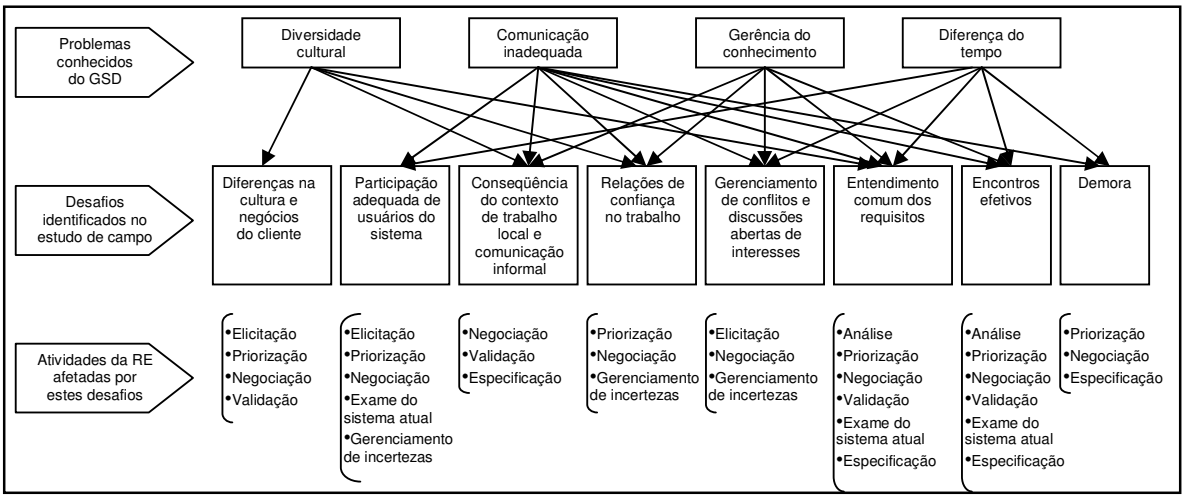


Figura 18 - Modelo de impacto dos desafios e atividades afetadas da ER devido a problemas com o DGS (adaptado de [DAM02]).

Esses problemas se alinharam com estudos anteriores de DGS [CAR99]. A segunda camada da figura mostra as dificuldades específicas encontradas, decorrentes dos problemas identificados. Na terceira camada, são apresentadas as atividades da ER afetadas por cada uma das dificuldades encontradas. O estudo, além da construção do modelo, pôde obter também diversos resultados, relacionados com conflito, cultura e distância.

Estes estudos sugerem que a distância tende a exacerbar os problemas fundamentais da ER, como a falta de comunicação entre *stakeholders*, bem como os fatores de natureza política, organizacional e social. Sugere também que pesquisas de campo devem ser realizadas para que se entenda o impacto dos problemas de comunicação e coordenação nas atividades de ER, identifique os desafios apresentados pelas organizações distribuídas, e apresente recomendações para superar os problemas associados com a distância.

## 3.3 Abordagem de Desouza e Evaristo para Gestão de Conhecimento

Desouza e Evaristo [DES04] apresentam em seu artigo uma abordagem hibrida para a gestão de conhecimento em projetos distribuídos. Através da análise do trabalho de Hansen e seus colaboradores [HAN99], que dividem as abordagens de gestão de conhecimento em codificação e personalização, os autores traçam um paralelo das mesmas com dois modelos computacionais muito populares: cliente-servidor e ponto-a-ponto (P2P), respectivamente.

De acordo com os autores, na abordagem de codificação o conhecimento individual é condensado, contextualizado e centralmente disponibilizado em banco de dados. Esta abordagem parte da premissa que o conhecimento pode ser facilmente extraído e codificado, tornando-a muito similar ao modelo cliente-servidor. Já a personalização é exatamente o oposto, pois reconhece a dimensão tácita do conhecimento e assume que o conhecimento é compartilhado principalmente pelo contato direto entre os indivíduos. Esta abordagem tem um caráter distribuído que a torna muito semelhante ao modelo P2P.

Para identificar quais informações beneficiam-se mais da codificação ou da personalização, os autores utilizam a classificação de conhecimento proposta por [DMM02]:

- Conhecimento em projetos;
- Conhecimento sobre projetos;
- Conhecimento proveniente de projetos.

As duas abordagens têm implicações na gestão de conhecimento em projetos distribuídos, levando os autores a analisar suas vantagens e desvantagens sobre três dimensões da gestão do conhecimento: compartilhamento, controle e estruturação do conhecimento. Com base na análise citada, Desouza e Evaristo propõem uma abordagem híbrida visando maximizar os benefícios dos modelos cliente-servidor e P2P, quando aplicados à gestão do conhecimento.

Tanto na gerência de requisitos quanto na gerência de projetos realizadas em ambientes distribuídos a gestão do conhecimento torna-se um fator chave para o sucesso. Estas atividades, quando realizadas em DDS, apresentam uma complexidade extra causada pela necessidade de gerenciar as interdependências entre os elementos distribuídos através do tempo, do espaço e de múltiplos projetos.

Nesta dissertação de mestrado a abordagem híbrida proposta por Desouza e Evaristo é adaptada e aplicada à gerência de requisitos em DDS. Vários dos desafios encontrados na ER em DDS podem ser tratados através desta abordagem. Mas possivelmente, o processo de gerência de requisitos seja o beneficiado pela adoção de práticas de gestão do conhecimento, dado que é o processo afetado pelas dificuldades evidenciadas pela manutenção de software, tal como apresentado na seção 2.4.2 e no capítulo 5.

# 4 Método de Pesquisa

Após revisão teórica, percebeu-se que o problema apresentado ainda não foi abordado sob a mesma perspectiva. Assim, esta pesquisa se caracteriza como um estudo predominantemente exploratório. Segundo Yin [YIN01] e Gil [GIL95], a pesquisa exploratória tem como principal finalidade desenvolver, esclarecer e modificar conceitos e idéias, com vistas à formulação de novas teorias, modelos e hipóteses pesquisáveis em estudos posteriores. Freqüentemente, o tema em foco é genérico, tornando-se necessário seu esclarecimento e delimitação, exigindo uma consistente revisão da literatura, discussão com especialistas e outros procedimentos.

A pesquisa exploratória muitas vezes constitui-se na primeira etapa de uma investigação mais ampla, que é o caso deste estudo. Neste estudo, pode-se justificar o uso de métodos qualitativos pelo fato de envolver o estudo da ER em ambientes DDS no seu contexto real, com a descrição e a compreensão do estado da arte naquelas situações em que a prática se antecipa à teoria [HOP97]. Os dois estudos de caso utilizados nesta pesquisa foram realizados em um centro de desenvolvimento de software de uma multinacional de grande porte.

## 4.1 Desenho de pesquisa

O desenho de pesquisa apresentado na Figura 19 apresenta as principais etapas desta pesquisa. A pesquisa foi organizada em três fases, descritas a seguir.

Na fase 1 foi realizado um extenso estudo da base teórica nos assuntos relacionados, como ER, DDS e manutenção de software. Esta fase foi fundamental para formação de uma base conceitual consistente para a continuidade do estudo. Ainda nesta fase foi realizado o estudo de caso 1. O estudo de caso 1 caracterizou-se por um estudo exploratório visando identificar o processo de ER seguido por uma organização em projetos de manutenção de software em ambientes de DDS e as principais dificuldades enfrentadas neste contexto. Os resultados deste estudo são apresentados no capítulo 5.

Na fase 2, após a consolidação de uma base teórica consistente no assunto e tendo como base os resultados obtidos no estudo de caso 1, foi elaborado um modelo de arquitetura de informação para gerência de requisitos em DDS. Este modelo é apresentado no capítulo 6 deste volume.

Na fase 3, foi realizado o estudo de caso 2, visando avaliar a aplicabilidade do modelo proposto e os benefícios decorrentes de sua utilização. Os resultados deste estudo são

apresentados no capítulo 7. Com base nos resultados obtidos no estudo de caso 2 o modelo de arquitetura de informação para gerência de requisitos em DDS foi refinado e a versão final, apresentada no capítulo 6, foi elaborada.

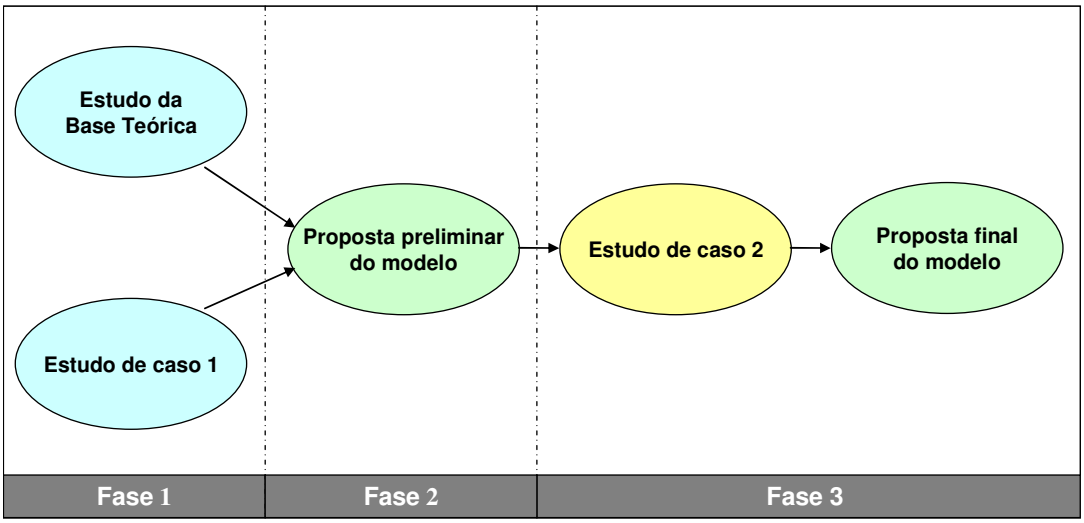


Figura 19 - Desenho de pesquisa.

Por tratar-se de uma pesquisa qualitativa, devem-se ter claras as limitações deste tipo de pesquisa, principalmente no que se refere ao número de projetos e respondentes, restringindo a generalização dos resultados obtidos.

## 4.2 Considerações sobre a evolução da pesquisa

O objetivo inicial da pesquisa apresentada nesta dissertação de mestrado era a elaboração de um processo de gerência de requisitos para manutenção de software em DDS. Após a primeira fase da pesquisa, com o amadurecimento da pesquisa, a participação em eventos da área e as contribuições do grupo de pesquisa em engenharia de software do PPGCC, algumas alterações no objetivo da pesquisa tornaram-se necessárias.

A primeira mudança consistiu da alteração do foco da pesquisa. Ao invés de restringir a proposta à fase de manutenção de software decidiu-se por estendê-la, passando a ser aplicável à qualquer fase do ciclo de vida de um produto de software. Assim, esta proposta foi elaborada para ser utilizada tanto em projetos de desenvolvimento de software quanto em projetos de manutenção de software. Esta decisão foi motivada pela percepção de que as dificuldades existentes na ER quando utilizada neste contexto não eram restritas a projetos de manutenção de software nem tão pouco causadas pelos mesmos, sendo apenas evidenciadas nesta etapa do ciclo de vida do produto de software. Percebeu-se que, geralmente, estas dificuldades tinham origem nos projetos de desenvolvimento de software. Não havendo, desta forma, motivos para limitar a solução apenas à fase de manutenção de software. Assim, assumiu-se a hipótese de que as

dificuldades relatadas são apenas evidenciadas durante os projetos de manutenção de software e não causadas pelos mesmos.

A segunda mudança consistiu da redução do objetivo da pesquisa. Ao invés da proposta de um processo passou-se a buscar a elaboração de um modelo de arquitetura de informação para gerência de requisitos em DDS. Esta decisão decorreu da percepção de que o objetivo inicial, de elaboração específica de um processo, poderia exigir um esforço de desenvolvimento e pesquisa incompatível com os recursos disponíveis e com o cronograma inicialmente estabelecido.

Estes ajustes no objetivo da pesquisa foram discutidos e avaliados com pesquisadores do PPGCC durante o seminário de andamento desta pesquisa de mestrado.

# 5 Estudo de Caso 1

O estudo de caso 1 teve como objetivo Identificar o processo de ER seguido pela organização em projetos de manutenção de software e os principais problemas enfrentados neste contexto. Com este propósito foram realizadas entrevistas com duas equipes envolvidas em projetos de manutenção de software em ambientes distribuídos. A seção 5.1 apresenta a descrição do estudo realizado. A seção 5.2 apresenta os elementos de análise. A seção 5.3 apresenta as considerações finais deste estudo de caso.

## 5.1 Descrição

Quanto ao processo de pesquisa, o estudo de caso foi realizado em três etapas. A primeira etapa foi de planejamento, onde foi desenvolvido o protocolo de estudo de caso e o roteiro de entrevista. Para isto foram realizadas três atividades. Primeiro foi realizada uma reunião entre o pesquisador e professor orientador para levantamento das questões e estruturação do roteiro de entrevistas. Em seguida foi realizada a validação de face e conteúdo do protocolo de estudo de caso, por um pesquisador sênior. Com base nesta validação o protocolo sofreu pequenas correções. A seguir foi realizado o pré-teste, no qual foram entrevistados dois funcionários da organização. Os dois funcionários entrevistados não participaram do estudo e suas entrevistas não foram consideradas durante a análise dos resultados. Com base neste pré-teste foi possível executar os últimos ajustes no roteiro de entrevistas. A versão final do protocolo de pesquisa contendo o roteiro de entrevistas e descrevendo todos os procedimentos seguidos encontra-se no Apêndice A deste volume.

A segunda etapa do estudo de caso consistiu da realização das entrevistas. Esta etapa foi realizada seguindo os procedimentos definidos durante a etapa de planejamento. Foram realizadas oito entrevistas ao longo de 15 dias, respeitando-se a disponibilidade de horários dos entrevistados. Todas as entrevistas foram gravadas em fita cassete e posteriormente digitalizadas e armazenadas em CD. Este procedimento visa tanto facilitar o acesso aleatório a trechos de entrevistas durante a análise quanto garantir a integridade do material armazenado para o caso destas informações tornarem-se necessários em estudos futuros.

A ultima etapa do estudo de caso consistiu da análise de conteúdo das entrevistas. Nesta etapa todas as entrevistas foram analisadas pelo pesquisador e um relatório preliminar foi apresentado ao orientador. Ao todo foram realizadas três reuniões entre o orientador e o

pesquisador para discussão dos resultados e das categorias selecionadas na análise de conteúdo. Como resultado destas reuniões foi redigido este relatório apresentando os resultados obtidos.

Cabe salientar aqui duas mudanças realizadas durante a análise dos resultados com relação ao que havia sido estabelecido no planejamento. A dimensão 5 definida no protocolo foi dividida, durante a análise, em duas seções. Isto se justifica pelo fato das questões abordarem tanto aspectos relacionados às dificuldades encontradas na realização da ER em projetos de manutenção quanto às soluções adotadas para minimizar os efeitos destas dificuldades. Já as dimensões 2 e 3 foram unidas, para fins de análise, em uma única seção. A dimensão 2 trata do processo de manutenção de software e a dimensão 3 da gerência dos projetos de manutenção de software, estando portanto fortemente relacionadas. Apesar dos procedimentos de validação adotados durante a etapa de planejamento, isto não foi detectada em tempo hábil, sendo alterado durante a análise. Entretanto, esta alteração não compromete o estudo, por tratar-se apenas de um aspecto de organização com a finalidade de facilitar a apresentação dos resultados, não alterando de forma alguma os procedimentos seguidos ou os resultados obtidos.

### 5.1.1 Caracterização da organização

O estudo de caso foi desenvolvido em uma unidade de desenvolvimento de software de uma organização de grande porte. A organização possui escritórios em mais de 34 países em todo o mundo, inclusive o Brasil. Segundo informação fornecida pela própria organização, possui em torno de 50.000 colaboradores em todo o mundo.

A unidade onde o estudo foi aplicado está localizada na cidade de Porto Alegre, estado do Rio Grande do Sul, Brasil. Ela possui mais de 400 colaboradores trabalhando em projetos que atendem as necessidades da área de TI da empresa. Atuando em um ambiente de DDS, a maior interação é com a matriz nos Estados Unidos, responsável pela demanda dos projetos. A unidade é certificada no nível dois de maturidade de desenvolvimento de software do padrão SW-CMM desde 2003.

A unidade estudada é coordenada por um diretor responsável por toda a parte administrativa e operacional da unidade (desde recursos humanos até o desenvolvimento dos projetos). Ele é o responsável pelo contato com os diretores da organização como um todo. Abaixo do diretor existe um departamento responsável pelo suporte administrativo e um consultor de RH. Existe ainda a área de qualidade, coordenada pelo *software engineering process group* (SEPG).

Com relação aos projetos, existe uma estrutura organizacional mista, com três áreas de desenvolvimento de software (corporativa, comercial e industrial). Cada área possui um gerente de desenvolvimento, chamado de *delivery manager* (DM), responsável pela alocação dos integrantes das equipes de projetos e responsável também pela gerência da equipe. Os DM's

também são responsáveis pela negociação dos projetos e pelo contato com os gerentes de conta e com os clientes. Além disto, cada projeto possui os seguintes integrantes: um *project manager* (PM), responsável pela gerência de projeto; um *system architect* (SA), responsável pela análise do problema e pela ER; pelo menos um *tech lead* (TL), responsável pela especificação da solução; e pelo menos um *test lead* (TSL), responsável pela especificação dos planos de teste, uma equipe de desenvolvedores e outra de testadores.

### 5.1.2 Caracterização dos projetos analisados

Nesta pesquisa foram analisados dois projetos, chamados doravante de Projeto 1 e Projeto 2. Os projetos tratam de manutenções corretivas, adaptativas e perfectivas em sistemas utilizados em várias unidades da organização.

Em cada projeto foram entrevistados quatro integrantes. No Projeto 1 foram entrevistados o DM, o PM, o TL e o TSL. Não havia, no Projeto 1, um integrante designado exclusivamente para a função de SA, mas esta função era desempenhada pela TL. Esta acumulação de funções foi atribuída pelo DM do Projeto 1 como sendo decorrente do porte do projeto, que não justificaria a alocação de um profissional exclusivamente para a função de SA. Já no Projeto 2, foram entrevistados o DM, o PM, o SA e o TSL. Neste caso não havia um integrante designado exclusivamente para a função de TL, mas esta função era desempenhada pelo SA do projeto, o qual, de acordo com o DM do Projeto 2, possuía a qualificação técnica e a experiência necessária para desempenhar as duas funções. Novamente, o fato foi justificado pelo porte do projeto, que não justificaria a alocação de um profissional exclusivamente para esta função.

O Projeto 1 consiste da manutenção em um sistema de comércio eletrônico utilizado pela organização. Enquanto que o Projeto 2 consiste da manutenção de um sistema da área financeira da organização. Ambos são sistemas legados, desenvolvidos na matriz da organização. Sendo que recentemente a manutenção destes sistemas foi delegada a unidade de desenvolvimento de software onde este estudo foi realizado.

Nos dois projetos a equipe de *stakeholders* está geograficamente distribuída, sendo composta por integrantes da matriz e de diversas unidades da organização no mundo. Já a equipe de mantenedores e de testadores do sistema encontra-se co-localizadas na unidade onde foi realizado este estudo.

### 5.1.3 Caracterização dos respondentes

As questões de 1 a 11 do roteiro de entrevistas visavam estabelecer a caracterização dos respondentes. Foram realizadas entrevistas com 8 profissionais envolvidos em dois projetos distintos. A intenção inicial era entrevistar 10 pessoas, mas nos dois projetos as funções de *System*

*Architect* e *Tech Lead* eram exercidas pela mesma pessoa. Os participantes foram selecionados em função de seu papel na organização. Foram entrevistados *Delivery Managers*, *System Architects*, *Project Managers*, *Tech Leads* e *Test Leads*.

A maioria dos entrevistados possui pelo menos 7 anos de experiência na área de Informática, sendo o tempo médio de experiência de 12,6 anos. A média de idade dos entrevistados é de 33,5 anos, sendo a idade mínima de 26 anos e a idade máxima de 44 anos. De todos os entrevistados, 75% possuem um tempo de atuação na organização entre 2 e 5 anos e o restante possui um tempo de atuação de até 18 meses.

As entrevistas tiveram uma duração média de 30,2 minutos (entre um mínimo de 15 minutos e um máximo de 41 minutos de duração) e contaram com total disponibilidade e atenção dos participantes. Foram fornecidas todas as informações solicitadas, sempre respeitando a política de privacidade e confidencialidade da organização.

Com relação ao nível de formação, o grupo representa adequadamente o alto nível de qualificação dos funcionários da organização, sendo que todos os respondentes possuem Curso Superior completo e 37,5% estão cursando mestrado. Com relação à formação acadêmica, 2 vêm das áreas da Ciência da Computação, 3 da Análise de Sistemas, 2 são Tecnólogos em Processamento de Dados e um é Engenheiro Mecânico. A função dos respondentes se distribui conforme a Figura 20.

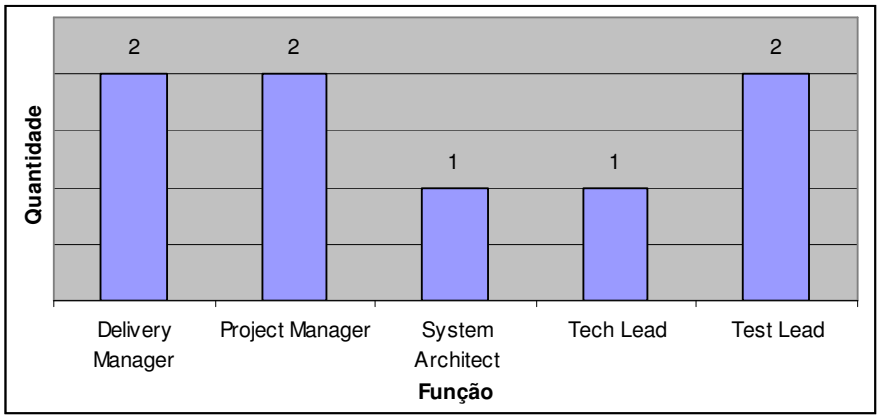


Figura 20 - Distribuição dos respondentes.

## 5.2 Elementos de análise

A principal contribuição deste estudo está na análise do processo ER em projetos de manutenção de software em ambientes de DDS. A seguir apresentam-se os elementos de análise e as categorias obtidas com a análise de conteúdo.

### 5.2.1 Processo de manutenção

As questões de 12 a 19 buscavam explorar a percepção dos respondentes com relação ao processo de manutenção de software utilizado pela organização. A questão 12 visava identificar se existia um processo definido para manutenção de software na empresa e como ele era chamado. Os resultados são apresentados na Tabela 4.

Tabela 4 – Processo de manutenção.

| **Existe um processo definido para manutenção de software?** | **Freqüência** |
|---|---|
| Sim. Adaptação do processo para manutenção de software. | 4 |
| Sim. Mas ainda estão trabalhando para adaptá-lo. | 1 |
| Não, é seguido o mesmo processo dos projetos novos. | 1 |

A maioria dos respondentes (4 respostas) respondeu que sim e que este processo é uma adaptação do processo de desenvolvimento utilizado pelo restante da unidade. Um dos respondentes ainda considera que o processo de adaptação ainda não está encerrado e que, portanto, não pode classificá-lo como "bem definido". Apenas um respondente acredita não existir um processo específico para manutenção de software.

Com base nas respostas da questão 13 foi possível obter uma descrição do processo de manutenção. Este processo é uma adaptação do processo de desenvolvimento de software da unidade analisada. A unidade possui um processo de desenvolvimento de software bem definido, seguindo as normas e padrões do modelo de maturidade SW-CMM (nível dois). O processo tem como base o MSF para o desenvolvimento de software e a metodologia do *Project Management Institute* (PMI) para o gerenciamento de projetos. Além disso, técnicas de desenvolvimento *Agile* também são utilizadas, dependendo das características dos projetos.

A diferença entre o processo de manutenção e o processo padrão de desenvolvimento de software se dá principalmente no ciclo de vida adotado. Existem projetos de manutenção com *releases* mensais e trimestrais desenvolvidos concorrentemente, sendo que os vários projetos de manutenção de um mesmo sistema fazem parte de um programa de manutenção. Todo o planejamento das manutenções de um sistema é feito no escopo do programa de manutenção, que tem duração de um ano, e é válido para todos os projetos que compõem o programa. Assim, espera-se reduzir o esforço necessário para manter a documentação de planejamento dos projetos (esforço chamado pelos respondentes de "*paperwork*") e manter o foco nas atividades de engenharia de software.

As solicitações de manutenção de maior urgência são tratadas nos *releases* mensais, sendo as demais atendidas pelos *releases* trimestrais. Cada requisição de manutenção é negociada com o analista de negócios responsável pelo sistema e agendada para o release mais adequado, de acordo com sua urgência. A única exceção se dá no atendimento de problemas críticos que possam afetar a produção da organização. Neste caso o problema recebe prioridade máxima e o

processo definido não é necessariamente seguido.

De acordo com todos os respondentes, o artefato recebido como entrada para o processo de manutenção (questão 14) é o *Software Requirements Specification* (SRS), que é usado tanto para a especificação de novos sistemas quanto para a especificação das manutenções. Apenas um dos respondentes acrescentou que existe uma exceção no caso das correções urgentes. Nestes casos não existe um formalismo estabelecido e as requisições podem chegar à unidade por diversos meios, principalmente por telefone e e-mail. Todos os respondentes afirmam que como artefato de saída do processo (questão 15) é entregue o sistema modificado e um documento descrevendo o processo de implantação da nova versão. Apenas um dos respondentes acrescentou que também é entregue o resultado dos testes realizados.

Os respondentes são unânimes em afirmam que a maior parte das manutenções solicitadas divide-se em manutenções adaptativas e perfectivas, que é feita pouca manutenção corretiva e nenhuma preventiva.

### 5.2.2 Engenharia de Requisitos

As questões de 20 a 28 buscavam explorar a percepção dos respondentes com relação ao processo de ER seguido nos projetos de manutenção de software em ambientes DDS realizados na unidade analisada.

Quando questionados se existia alguma técnica formal de ER sendo utilizada, os respondentes foram unânimes ao afirmarem que não seguiam nenhum processo ou técnica formal de ER. Sendo que um dos *delivery manager* afirmou que já houve tentativas de usar Casos de Uso para especificar os requisitos, mas que geralmente a equipe já recebe os requisitos prontos em um nível de detalhe muito técnico, quase como um português estruturado, inviabilizando, na sua opinião, o uso desta técnica. Este respondente acredita que o problema pode estar relacionado ao fato de os profissionais de plataforma de grande porte, envolvidos no projeto, terem um perfil muito técnico, tanto na unidade analisada quanto na matriz da organização nos EUA.

Os respondentes também foram unânimes ao afirmarem que não recebem nenhuma documentação referente aos requisitos anteriores do software. O único artefato recebido como entrada para o processo de ER nos projetos analisados é o SRS com a especificação de requisitos para o projeto atual. Afirmam também que não são documentadas as alterações em requisitos anteriores do produto que é submetido à manutenção, não havendo, portanto, informações que permitam o rastreamento entre os requisitos antigos e novos ou entre os requisitos e as manutenções que os modificam.

Este aspecto do processo de ER aplicado a projetos de manutenção em DDS é importante, pois, de acordo com [KOT98], o rastreamento entre os requisitos é um aspecto crítico do processo de gerência dos requisitos, pois é necessário para viabilizar a análise de impacto que

uma mudança nos requisitos pode acarretar no sistema. Além disto, se considerarmos a especificação de requisitos como parte da documentação de um sistema computacional, então este aspecto está diretamente relacionado à ausência de documentação adequada dos sistemas submetidos à manutenção. Esta questão e sua relação com DDS são analisadas em maior profundidade na seção 5.2.3, que trata das dificuldades encontradas nos projetos de manutenção em ambientes de DDS.

A influência de DDS também pode ser observada na questão relativa ao conceito de requisito, como demonstrado na Tabela 5.

Tabela 5 – Definição de requisito.

| **Definição de requisito** | **Respostas** | **Função do respondente** |
|---|---|---|
| Necessidade ou Característica do produto | 5 | Test Lead, Tech Lead e DM |
| Tarefa a ser realizada no projeto | 3 | PM e SA |

A maioria dos respondentes acredita em um conceito de requisito condizente com a literatura, especialmente com a definição de [THA90], onde requisitos são características do software necessárias para o usuário solucionar um problema de forma a atingir um objetivo ou ainda uma característica do software que deve ser realizada ou possuída por um sistema ou componente de sistema para satisfazer um contrato, padrão, especificação ou outra documentação formalmente imposta.

Entretanto, uma observação importante é de que os dois *project managers* e o *system architect* definiram requisitos como tarefas a serem realizadas por uma equipe de manutenção, conceito não respaldado na literatura sobre o tema. Este aspecto chama atenção, pois estes respondentes, apesar de não representarem uma maioria, exercem papeis importantes nos projetos. Um trecho da entrevista com o *system architect*, papel responsável pelas atividades de ER nos projetos conduzidos pela organização, ajuda a ilustrar este aspecto. Quando questionado a respeito do que é, em sua opinião, um requisito, o *system architect* respondeu:

> *"Requisito é uma solicitação... uma feature... algo que tem que ser feito."*

Também pode ser destacada a resposta de um dos *project manager*, quando questionado a respeito do que é, em sua opinião, um requisito:

> *"Um statement do que tem que ser feito em termos de melhorar ou corrigir algum defeito no software."*

As duas respostas apresentam uma visão de que um requisito é "algo a ser feito", ou seja, uma tarefa definida e que deve ser cumprida durante o projeto. Esta visão pode estar relacionada à maneira como as atividades de ER estão distribuídas, em virtude do ambiente de DDS adotado pela organização. Neste ambiente as responsabilidades pela execução das atividades do processo de ER são distribuídas entre a equipe de desenvolvimento local, da qual o *project manager* e o

*system architect* fazem parte, e a equipe de usuários remota, que está localizada nos EUA. Como a especificação de requisitos é elaborada pela equipe remota, os respondentes acabam tendo um baixo envolvimento com a atividade de elicitação de requisitos. Isto faz com que recebam apenas as tarefas que devem ser realizadas para atender os requisitos, ao invés dos requisitos propriamente ditos, podendo assim ter sua opinião sobre o que são requisitos influenciada pela especificidade do contexto em que trabalham.

Quando questionados sobre as atividades do processo de ER em que participavam, a maioria dos respondentes afirmou participar apenas da análise e negociação e da validação dos requisitos. A Figura 21 demonstra os resultados obtidos.

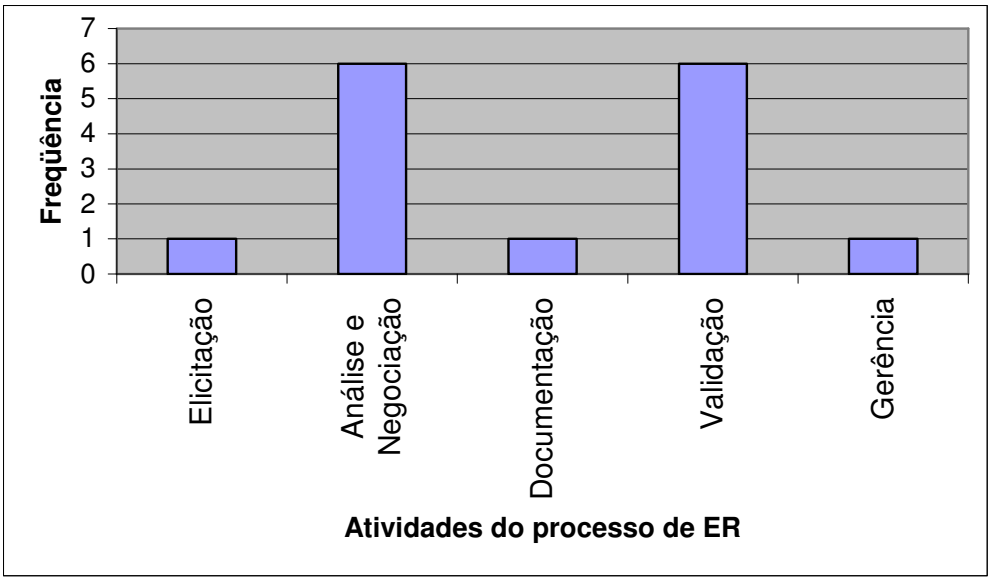


Figura 21 - Participação dos respondentes no processo de ER.

Apenas o *system architect* afirma participar da elicitação de requisitos. Os demais respondentes afirmam que a atividade de elicitação de requisitos é realizada pelos clientes e que a equipe de manutenção tem muito pouca influência sobre estas atividades, restando apenas a possibilidade de analisar, negociar e validar os requisitos. Apenas um dos *project manager* afirma participar da documentação e da gerência dos requisitos. Os demais respondentes alegam não ter envolvimento com estas atividades.

O trecho de entrevista transcrito a seguir ajuda a ilustrar a opinião do *system architect* sobre sua participação no processo de engenharia de requisito:

> *"A gente trabalha aqui já recebendo o SRS (software requiremetns specification). Eu recebo o SRS, que no caso é feito por uma pessoa de IT da matriz (nos EUA) que trabalha com o usuário. Ai eu reviso este SRS e com isto eu vou atrás dos programas, olho os programas, olho o sistema... Ai eu desenvolvo a especificação do software, e existe uma discussão se isto deve ser feito pelo tech lead ou pelo system architect.*

> *Mas a parte de requisitos de negócio deveria ser feita pelo system architect e a especificação técnica seria o tech lead. Só que nesta plataforma tu recebe os requisitos já bem técnicos. Eu já recebo no SRS: 'mexer no programa tal'. Então eu passo isto para a especificação, mas não tem muito mais para acrescentar. Mas eu faço mais um refinamento para verificar qual a subrotina que eu preciso tratar... este tipo de coisa.".*

Analisando-se o conjunto de atividades do processo de ER realizadas pelos respondentes e a forma distribuída como as demais atividades são realizadas é possível perceber a influência de DDS sobre o processo de ER. De acordo com [KOT98], as atividades de elicitação, análise, negociação, documentação, validação e gerência de requisitos deveriam ser realizadas pelo engenheiro de requisitos, função exercida na organização pelo *system architect*. Entretanto o autor não trata do processo de ER aplicado a ambientes de DDS, limitando sua abordagem a projetos de desenvolvimento de software em ambientes onde os *stakeholders* estão co-localizados.

Portanto, os resultados encontrados com a análise do processo de ER reforçam a importância da adoção de abordagens de ER para ambientes de DDS, tal como proposto por [LOP04], que define um modelo de processo de ER para ambientes de DDS que distribuiu as responsabilidades entre um time de especificação, remotamente localizado, e outro local de desenvolvimento, visando minimizar as dificuldades enfrentadas neste cenário.

### 5.2.3 Dificuldades

As questões 29 e 30 buscavam explorar a percepção dos respondentes com relação às principais dificuldades vivenciadas na ER em projetos de manutenção de software em ambientes de DDS. Após a análise das respostas obtidas e as discussões entre os pesquisadores envolvidos (orientador e pesquisador), ficaram definidas as seguintes categorias (Tabela 6):

Tabela 6 – Dificuldades encontradas.

| **Dificuldade** | **Freqüência** |
|---|---|
| Ausência de documentação adequada | 5 |
| Ausência de conhecimento sobre a aplicação | 4 |
| Falta de metodologia ou padrão para escrita de requisitos | 4 |
| Falta de planejamento | 1 |

Quando questionados sobre as atividades onde estas dificuldades foram encontradas, a maioria dos respondentes respondeu que as dificuldades relacionadas a requisitos surgiam em todas as atividades da manutenção. A Tabela 7 apresenta as categorias encontradas.

Tabela 7 – Atividades afetadas.

| Atividades | Freqüência |
|---|---|
| Todas | 5 |
| Testes | 2 |
| Elicitação de requisitos | 1 |

A principal dificuldade apontada pelos respondentes, a ausência de documentação adequada, é apontada por diversos autores ([EBN02][LIU99][PRE01][WHI95][ZAN03]) como a principal dificuldade encontrada em projetos de manutenção de sistemas legados. Os mesmos autores também afirmam que muitas vezes os *stakeholders* do projeto original também não estão mais na organização. Estes fatores fazem com que, em muitas circunstâncias, o conhecimento preciso sobre o sistema seja perdido. Nestes casos o sistema em si, seus códigos-fonte e aqueles que o usam são as únicas fontes de informação sobre o que o sistema faz, como faz e porque faz.

Entretanto, o ambiente de DDS utilizado na unidade agrega novos elementos a esta dificuldade, pois esta situação também poderia estar relacionada à dificuldade de acesso à documentação, causada pela distribuição geográfica entre a equipe de mantenedores e os demais *stakeholders* do projeto, localizados na matriz, onde os sistemas foram originalmente desenvolvidos. Este fator agrega dificuldades de comunicação que podem maximizar as dificuldades comumente encontradas em projetos de manutenção. Devido ao escopo do estudo realizado, que limitou-se a unidade de desenvolvimento de software localizada no Brasil, não foi possível determinar se a documentação necessária para os projetos de manutenção realmente não existe ou apenas não é enviada pela matriz da organização para a equipe de mantenedores.

A ausência de conhecimento sobre a aplicação, segunda dificuldade apontada pelos respondentes, remete a importância da gestão do conhecimento para o processo de ER em ambientes de DDS. De acordo com [DAM02] e [LOP03], a gestão de conhecimento influencia a ER em ambientes DDS. O processo ER em DDS envolve, por exemplo, documentos cujo conteúdo pode ter diversas origens. Fatores como diferenças culturais e de idioma dificultam o processo de transferência de conhecimento sobre a aplicação entre os diversos *stakeholders* geograficamente distribuídos. Além disto, sistemas legados costumam ser utilizados e mantidos por longos períodos de tempo. Isto faz com que, até mesmo em organizações com baixo *turn over*, seja alto o risco de que as pessoas que detêm o conhecimento sobre o sistema deixem a organização ao passar dos anos, fazendo com que este conhecimento seja gradualmente perdido pela organização. Captar, processar, armazenar e disponibilizar globalmente o conhecimento gerado e consumido pelos processos de manutenção e ER são questões que podem ser endereçadas pela gestão do conhecimento.

A ausência de documentação aliada ao baixo nível de conhecimento sobre a aplicação acaba criando na equipe de mantenedores uma sensação de insegurança quanto aos possíveis efeitos colaterais que uma manutenção pode provocar. Esta sensação, observada em alguns dos

respondentes, demonstra porque a manutenção de software já foi comparada por [CAN72] a um Iceberg, onde se espera que aquilo que está imediatamente visível seja tudo, mas sabe-se que uma enorme massa de custos e problemas esconde-se sob a superfície. Um trecho da entrevista transcrita de um dos *Test Leads* permite demonstrar este aspecto:

> *"Nós não recebemos os requisitos anteriores do sistema. Nós temos uma base onde guardamos as informações sobre o programa onde a gente está trabalhando em cima... da manutenção que está sendo feita. Mas eu acho que este aqui é o problema maior que a gente tem: quando esta fazendo a manutenção, a gente acaba conhecendo muito bem aquela partezinha em que tem que fazer a manutenção e acaba pecando por alterar alguma coisa sem fazer idéia de onde mais ela pode afetar. Já aconteceu de o sistema ir para produção e a gente ter que tirar do ar, pois não tinha esta visão de em qual módulo ela (a manutenção) poderia interferir. Eu acho que recebendo os requisitos anteriores a gente ia pelo menos enxergar com o que a manutenção interage. Só a manutenção não basta para ter uma visão do todo."*

Outra dificuldade citada pelos respondentes foi falta de metodologia ou padrão para escrita de requisitos em projetos de manutenção. Os requisitos atualmente são escritos em linguagem natural, sem o uso de nenhuma técnica formal de especificação. Também não existe nenhum padrão para o nível de abstração utilizado na escrita dos requisitos entre os diversos projetos. Alguns documentos apresentam os requisitos em um nível de regras de negócio, enquanto outros em um nível bastante técnico. Esta disparidade dificulta, principalmente, a criação de métricas de qualidade para os requisitos, pois inviabiliza comparações entre documentos escritos em níveis de abstração diferentes. Além disto, o uso de texto em linguagem natural também aumenta a ambigüidade dos requisitos, dificultando a interpretação das reais necessidades dos usuários.

Aparentemente, os relatos desta dificuldade foram estimulados por um trabalho que vem sendo realizado nos projetos de desenvolvimento de novos sistemas. A unidade esta adotando nestes projetos um modelo formal de especificação de requisitos em linguagem natural que visa minimizar as dificuldades encontradas na especificação de requisitos em ambientes de DDS. Mas este modelo ainda não foi aplicado em projetos de manutenção de software. Todos os respondentes que citaram dificuldades com a falta de metodologia ou padrão para escrita de requisitos também fizeram referências a este novo modelo como contra-exemplo.

Apenas um dos respondentes citou falta de planejamento como uma dificuldade enfrentada no processo de ER. Este respondente acredita que os problemas poderiam ser minimizados se houvesse mais planejamento para endereçar os requisitos e as modificações nos requisitos.

### 5.2.4 Soluções adotadas

As questões de 31 a 33 buscavam explorar a percepção dos entrevistados com relação às soluções adotadas até o momento para reduzir as dificuldades apontadas. Após a análise das respostas obtidas e as discussões entre os pesquisadores envolvidos (orientador e pesquisador), ficaram definidas as seguintes categorias (Tabela 8):

Tabela 8 – Soluções adotadas.

| Soluções adotadas | Freqüência |
|---|---|
| Maior interação com o cliente | 2 |
| Alocar profissionais experientes | 2 |
| Planejamento | 1 |
| Prototipação | 1 |
| Controle de qualidade dos requisitos | 1 |
| Geração da documentação | 1 |
| Geração de base de conhecimento | 1 |

Também foi realizado um mapeamento entre as soluções que cada respondente relatou e as dificuldades que o mesmo respondente havia citado. Este mapeamento é apresentado na Figura 22.

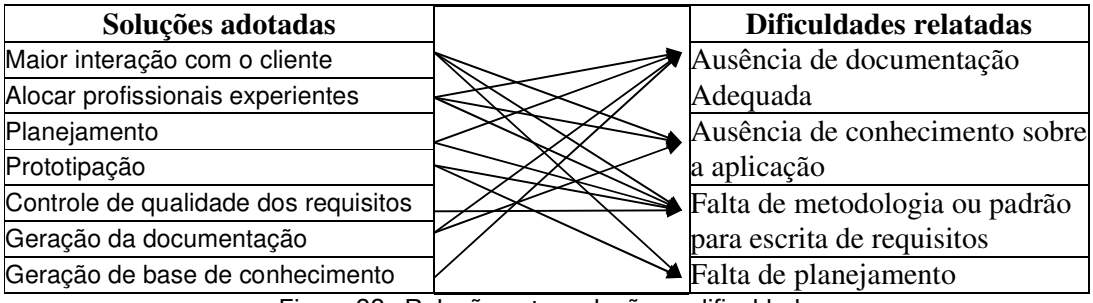


Figura 22 - Relação entre soluções e dificuldades.

Esta relação não é direta, pois alguns respondentes citaram mais de uma solução para uma dificuldade e outros apenas uma solução para várias dificuldades. É possível observar também que nem todas as soluções estão diretamente focadas nas dificuldades relatadas. Por exemplo, o respondente que citou a falta de planejamento como uma dificuldade não foi o mesmo que citou o planejamento como solução. Sendo que o planejamento foi citado como solução para ausência de documentação e para a falta de metodologia ou padrão para escrita de requisitos.

Quando questionados sobre a forma como estas soluções foram identificadas, a maioria dos respondentes afirmou que não foi utilizado método algum. As soluções citadas teriam sido criadas de maneira Ad Hoc, com base na experiência dos profissionais envolvidos. Um dos respondentes afirmou ter se inspirado nas técnicas de prototipação e um outro nas ferramentas que já vinham sendo utilizadas antes para armazenar os documentos do projeto. Estes resultados são apresentados na Tabela 9.

Tabela 9 – Como foram identificadas as soluções.

| Como foram identificadas as soluções | Freqüência |
|---|---|
| Ad hoc | 6 |
| Inspirados em técnicas de prototipação | 1 |
| Usando as ferramentas já existentes | 1 |

Quando questionados sobre o impacto que a solução adotada teve sobre o projeto e a organização, metade dos respondentes considerou os resultados positivos. A Tabela 10 apresenta os resultados obtidos.

Tabela 10 – Impacto das soluções adotadas.

| Impacto | Freqüência |
|---|---|
| Positivo | 4 |
| Negativo | 2 |
| Não soube avaliar | 2 |

Dois dos respondentes consideraram os resultados negativos. Um deles considera que depender da ajuda de profissionais mais experientes é uma solução que causa baixa produtividade, pois muitas das informações que necessita estão na memória destes profissionais, dificultando sua obtenção. O outro acredita que o uso de técnicas de prototipação acarreta outras dificuldades, pois quando o protótipo é apresentado o cliente tem dificuldade em compreender que o protótipo não é a versão final da solução oferecida e passa a querer mais modificações. Fazendo com que a equipe de mantenedores tenha que dedicar mais esforços na negociação com o cliente para evitar desvios desnecessários do plano do projeto ou crescimento do escopo. Os demais respondentes não souberam avaliar o impacto das soluções adotadas.

## 5.3 Considerações finais

Através da análise de conteúdo das entrevistas realizadas durante este estudo de caso foi possível obter informações relevantes quanto ao processo de ER em projetos de manutenção de software em ambientes de DDS e as principais dificuldades enfrentadas pelos respondentes. Além das categorias identificadas durante a análise isolada de cada elemento de estudo, também foi possível obter uma visão geral do problema.

O aspecto mais importante observado durante o estudo está relacionado ao processo de gerência dos requisitos nos projetos de manutenção. O estudo realizado permitiu identificar uma grande dificuldade na realização destas atividades, principalmente no que se refere ao rastreamento dos requisitos (*traceability*). Esta dificuldade esta relacionada em parte com a ausência de documentação adequada que permita a recuperação dos requisitos originais dos sistemas submetidos à manutenção e o correto registro das mudanças realizadas nestes

requisitos. Sendo que as dificuldades relativas à documentação do sistema podem ser exacerbadas no ambiente DDS em que os projetos são realizados. Pois mesmo que a documentação contendo os requisitos do sistema exista, pode estar em outro site de desenvolvimento, ou escrita em outro idioma e em outro contexto. Dificultando assim, que as informações de rastreamento sejam mantidas. Além disto, em DDS os requisitos precisam ser criados ou atualizados em diferentes sites, por diferentes *stakeholders*. Para isto é necessário que o processo de gerência de requisitos leve em conta estratégias de gerência de configuração que permitam que *stakeholders* geograficamente dispersos possam realizar ou solicitar suas modificações nos requisitos.

Entretanto, os projetos de manutenção analisados não são os primeiros a serem realizados pela organização nos sistemas em questão. Mesmo assim, verificou-se que nem mesmo os requisitos trabalhados nos projetos anteriores, realizados na própria organização, foram submetidos a processos formais de gerência de requisitos que permitissem um trabalho continuado ao longo dos diversos projetos. A cada projeto de manutenção de um mesmo sistema são gerados novos artefatos de especificação de requisitos (SRS), sem que haja um gerenciamento das dependências entre os requisitos estabelecidos em cada projeto. Também não é feito o registro de informações de rastreamento das alterações de requisitos realizadas a cada projeto.

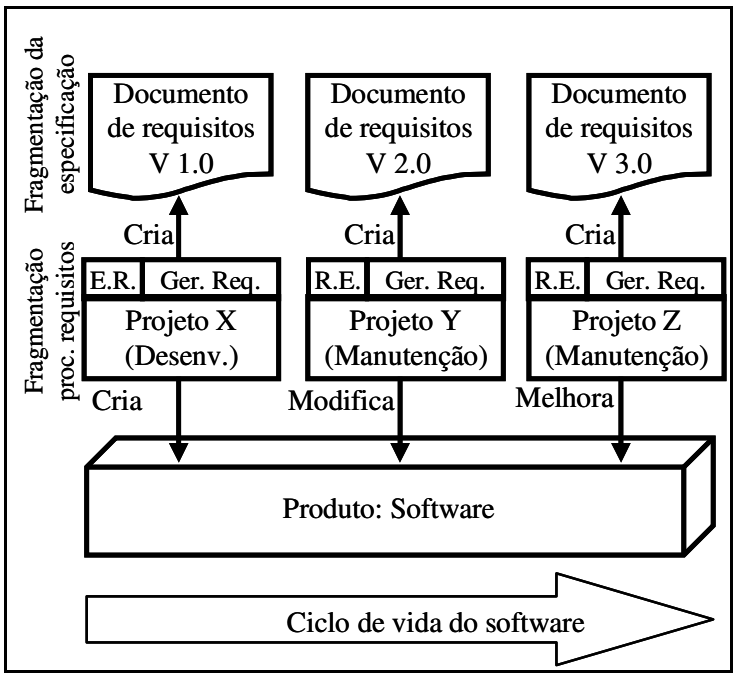


Figura 23 - Fragmentação das especificações de requisitos.

Esta fragmentação das especificações de requisitos, ilustrada pela Figura 23, contribui para as dificuldades de obtenção de conhecimento da aplicação legada, tal como apontado pelos respondentes e apresentado na seção 5.2.3. Também pode-se relacionar este problema com os resultados apresentados na seção 5.2.2, que demonstra um baixo envolvimento da equipe de manutenção com a gerência de requisitos. Apenas um dos respondentes, um *project manager*,

afirma estar envolvido nesta atividade. E nenhum dos *system architects*, que teoricamente deveriam ser os responsáveis por esta tarefa, respondeu ter envolvimento nesta atividade.

Estes fatores sugerem a necessidade de pesquisas adicionais que permitam desenvolver um processo de ER que contemple as necessidades existentes em projetos de manutenção de sistemas legados em ambientes DDS. Principalmente no que se refere ao processo de gerência de requisitos. Mesmo com a pesquisa bibliográfica realizada, não foram encontradas na literatura contribuições relevantes que permitam solucionar as dificuldades identificadas neste estudo de caso. As contribuições que mais se aproximam do contexto deste estudo limitam-se ao processo de ER em ambientes DDS em projetos de desenvolvimento de software, carecendo de mecanismos que lidem com a realidade encontrada na manutenção de software.

# 6 Modelo Proposto

Visando contribuir para a solução das dificuldades identificadas no estudo de caso 1 e na revisão bibliográfica realizada, este capítulo apresenta uma proposta de modelo de arquitetura de informação para gerência de requisitos em DDS. A idéia básica por trás desta arquitetura consiste em propor uma estrutura que permita a gerência de requisitos de forma integrada ao longo de todo o ciclo de vida do software e também em diferentes localidades, evitando assim a fragmentação das especificações de requisitos ao longo de diversos projetos e em diferentes localidades geograficamente distribuídas.

Apesar de este estudo ter sido motivado pelas dificuldades para ER em manutenção de software em DDS, decidiu-se por não restringir a solução proposta a este contexto. Esta decisão decorreu da percepção de que as dificuldades citadas não eram restritas a projetos de manutenção de software nem tão pouco causadas pelos mesmos, sendo apenas evidenciadas nesta etapa do ciclo de vida do produto de software. Percebeu-se que, geralmente, estas dificuldades tinham origem nos projetos de desenvolvimento de software. Não havendo, desta forma, motivos para restringir a solução apenas a projetos de manutenção de software. Sendo que a aplicação da solução poderia ter, inclusive, menor custo e maior eficiência quando aplicada desde o início do desenvolvimento do produto de software. Em virtude disto, o escopo do modelo deixou de estar limitado a projetos de manutenção de software, passando a ser aplicável a qualquer fase do ciclo de vida de um produto de software. Ao se verificar a literatura sobre ER, identifica-se que [ZAN03] reforça esta decisão, quando destaca que o conhecimento dos requisitos requer um esforço contínuo no refinamento progressivo das exigências contidas nas regras de negócio das organizações e em atendimento às necessidades dos *stakeholders*, durante todo o ciclo de vida do software.

A seção 6.1 apresenta as principais informações e artefatos envolvidos na ER. Este conjunto de informações será novamente abordado na descrição do modelo proposto. A seção 6.2 apresenta uma classificação do conhecimento em gerência de requisitos. Esta classificação é a base para determinar-se como cada uma das informações é gerenciada no modelo proposto. A seção 6.3 apresenta as implicações de cada uma das abordagens tradicionais de gestão do conhecimento quando aplicadas na gerência de requisitos em DDS. Por fim, a seção 6.4 descreve a arquitetura de informação para gerência de requisitos em DDS.

## 6.1 Principais informações e artefatos envolvidos na ER

A ER, como fase inicial do processo de desenvolvimento de software, envolve diversas formas de conhecimento, influenciados por fatores humanos, sociais e organizacionais. O processo de ER, além de garantir o desenvolvimento e aprovação de um artefato de requisitos, comumente chamado de SRS, envolve informações de diversas origens e propósitos, necessitando gerenciar as necessidades e expectativas dos envolvidos de forma a produzir um software adequado.

Segundo Kotonya [KOT98], as principais entradas e saídas do processo de ER são:

- Informações sobre o sistema existente;
- As necessidades dos *stakeholders*;
- Padrões organizacionais;
- Regulamentos, como de segurança e saúde;
- Informações sobre o domínio da aplicação;
- Requisitos acordados pelos *stakeholders*;
- Especificação do sistema;
- Modelos do sistema.

Ainda são considerados, durante a gerência de requisitos os artefatos necessários às atividades de controle de alterações e avaliação do impacto das mudanças, incluindo solicitações de mudança, aprovações, avaliação do impacto, rastreabilidade dos requisitos, entre outros.

Dependendo do processo de desenvolvimento de software utilizado, estas informações podem ser documentadas de formas distintas. O RUP [KRU00], por exemplo, utiliza como artefatos em seu *workflow* de gerenciamento de requisitos o documento visão (*Vision*), o modelo de casos de uso (*Use Case Model*) e a especificação suplementar (*Supplementary Specification*). Estes dois últimos documentos formam a especificação dos requisitos do software, na forma de um SRS. De forma similar, outros processos de software tendem a seguir o padrão da IEEE [IEE98], e utilizar um SRS. Além disso, para controle de alterações, o uso de um documento de requisição de mudanças (*Change Request*) é implementado por diversos processos.

## 6.2 Classificação do Conhecimento em Gerência de Requisitos

De acordo com [DMM02], o conhecimento gerado por projetos pode ser categorizado como:

- **Conhecimento em Projetos:** Compreende informações detalhadas que são geradas e utilizadas em cada projeto individual. Os envolvidos no projeto necessitam destas informações para saber quando, o que, quem, como, onde e por

que algo é feito, visando promover uma coordenação eficiente e efetiva das atividades.

- **Conhecimento sobre Projetos:** Compreende informações que a organização necessita para saber que projetos estão sendo conduzidos em qualquer momento. Estas informações contribuem para o planejamento e controle dos recursos utilizados nestes projetos.
- **Conhecimento Proveniente de Projetos:** Compreende as principais informações geradas com a execução do projeto. Este conhecimento é determinante do sucesso futuro do projeto e contribui para o aprendizado organizacional.

O primeiro passo na aplicação da abordagem híbrida de gestão de conhecimento à gerência de requisitos em DDS consiste em aplicar esta categorização às informações relevantes para ER e, em particular, para a gerência de requisitos. Desta forma é possível elaborar uma lista de informações para cada categoria. Durante a execução de um projeto, são necessárias várias informações detalhadas sobre os requisitos e as requisições dos *stakeholders*. Dentre estas informações, as seguintes podem ser destacadas como **conhecimento em projeto**:

- Novos requisitos e seus respectivos estados, tais como "proposto", "aprovado" ou "rejeitado";
- Requisições de alteração de requisitos e seus respectivos estados;
- Relatórios de análise de impacto das alterações requisitadas;
- Estimativas de custo e esforço para novos requisitos ou requisições de alteração em requisitos já existentes;
- Informações sobre a priorização dos requisitos;
- Controle de versão e controle de alteração de todos os requisitos do software.

De uma perspectiva mais ampla, a organização necessita de várias informações sobre a alocação de suas necessidades em requisitos de software e o andamento dos respectivos projetos. Estas informações são úteis para garantir o alinhamento dos esforços de TI com as estratégias da organização, contribuindo assim para o planejamento e controle dos recursos de TI. Sob esta perspectiva podemos destacar as seguintes informações como **conhecimento sobre projetos**:

- Projetos em andamento e as necessidades organizacionais sendo tratadas pelos mesmos;
- Produtos de software sendo criados ou modificados por cada projeto;
- *Stakeholders* envolvidos em cada projeto;
- Localidades (*sites*) envolvidas em cada projeto;
- Indivíduo ou equipe responsável por requisitos em cada projeto;
- Métricas relacionadas aos requisitos.

Já o **conhecimento proveniente de projetos** inclui as informações geradas pelos processos de ER após a execução dos projetos. Muitas informações evoluem durante a execução destes processos e atingem um estado estável e definitivo somente ao término do projeto, quando os requisitos já estão implementados e o processo de gerência de alterações já está encerrado. Nesta categoria podemos destacar:

- Produtos de software criados ou modificados pelo projeto;
- *Features* implementadas;
- Versão final dos requisitos;
- Informações de rastreabilidade.

A partir desta categorização é possível analisar as implicações de duas abordagens de gestão de conhecimento para cada categoria de conhecimento, bem como demonstrar as vantagens da abordagem híbrida.

## 6.3 Implicações das Abordagens de Gestão do Conhecimento para Gerência de Requisitos

Em [DES04], através da análise do trabalho de Hansen e seus colaboradores [HAN99], que dividem as abordagens de gestão de conhecimento em codificação e personalização, os autores traçam um paralelo das mesmas com dois modelos computacionais muito populares: cliente-servidor e ponto-a-ponto (P2P), respectivamente.

De acordo com os autores, na abordagem de codificação o conhecimento individual é condensado, contextualizado e centralmente disponibilizado em banco de dados. Esta abordagem parte da premissa que o conhecimento pode ser facilmente extraído e codificado, tornando-a muito similar ao modelo cliente-servidor. Já a personalização é exatamente o oposto, pois reconhece a dimensão tácita do conhecimento e assume que o conhecimento é compartilhado principalmente pelo contato direto entre os indivíduos. Esta abordagem tem um caráter distribuído que a torna muito semelhante ao modelo P2P.

As abordagens cliente-servidor e P2P apresentam vantagens e desvantagens de acordo com a categoria de conhecimento sendo considerada. Para avaliar qual abordagem oferece mais benefícios em cada categoria de conhecimento os autores da abordagem híbrida analisaram três dimensões da gestão de conhecimento: compartilhamento, controle e estruturação do conhecimento. Aqui, esta mesma análise é realizada para o contexto de ER em DDS.

### 6.3.1 Abordagem cliente-servidor

Na abordagem cliente-servidor, o conhecimento é codificado e centralmente

disponibilizado, garantindo assim uma estruturação do conhecimento proveniente de múltiplos projetos. Os produtores do conhecimento precisam organizar e classificar o conhecimento de acordo com padrões previamente estabelecidos pela organização. Assim é possível oferecer recursos de busca baseada em filtros que permitem um acesso rápido ao conhecimento desejado. A Figura 24 ilustra esta abordagem.

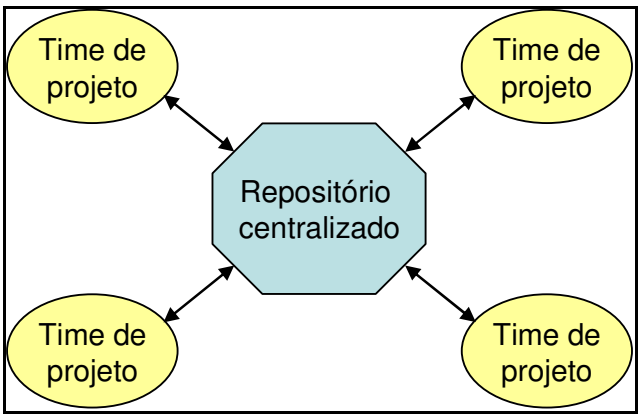


Figura 24 - Abordagem cliente-servidor (adaptado de [DES04]).

Do ponto de vista de ER em DDS, esta abordagem traz a vantagem de facilitar a transferência de conhecimento entre equipes geograficamente dispersas, pois garante que um mesmo padrão será utilizado para estruturar o conhecimento sobre requisitos de maneira uniforme por todas as equipes. Entretanto, esta abordagem pode dificultar o compartilhamento e o controle do conhecimento. Uma vez que o conhecimento precisa ser primeiro codificado de acordo com os padrões estabelecidos, os indivíduos tendem a retardar sua publicação até que tenham a confirmação e a estabilização do mesmo. Além disto, os indivíduos terão pouco controle sobre a visibilidade das informações a partir da sua publicação. Assim, uma parte do conhecimento sujeita a modificação mais freqüentes pode não ser disponibilizada com eficiência. Informações a respeito de requisições de alteração de requisitos e seu processo de avaliação, por exemplo, podem ser relegadas deixando-se que apenas as versões finais dos requisitos para publicação.

Desta forma, esta abordagem torna-se adequada para a gestão do conhecimento sobre projetos e proveniente de projetos, contribuindo para a preservação do conhecimento de longo prazo e para o aprendizado organizacional sobre as aplicações legadas e seus requisitos. Mas não é a opção mais adequada para o conhecimento em projetos, devido à natureza dinâmica deste conhecimento e às necessidades de controle e personalização existentes em cada projeto.

### 6.3.2 Abordagem P2P

Por outro lado, na abordagem P2P o conhecimento é personalizado. Os indivíduos podem escolher seus próprios meios de categorização e codificação do conhecimento e o

compartilhamento deste conhecimento se dá pelo contato direto entre os indivíduos. Neste caso, o papel da tecnologia é oferecer os meios necessários para facilitar a comunicação direta entre os indivíduos. Uma vez que os indivíduos são mais propensos a armazenar nos seus próprios repositórios as informações nas quais ainda estão trabalhando, esta abordagem pode reduzir os atrasos entre a criação do conhecimento e seu armazenamento no repositório. Ao contrario da centralização, onde o conhecimento é separado de seu criador, na abordagem P2P cada indivíduo possui controle sobre o conhecimento e sobre a visibilidade do mesmo. A Figura 25 ilustra esta abordagem.

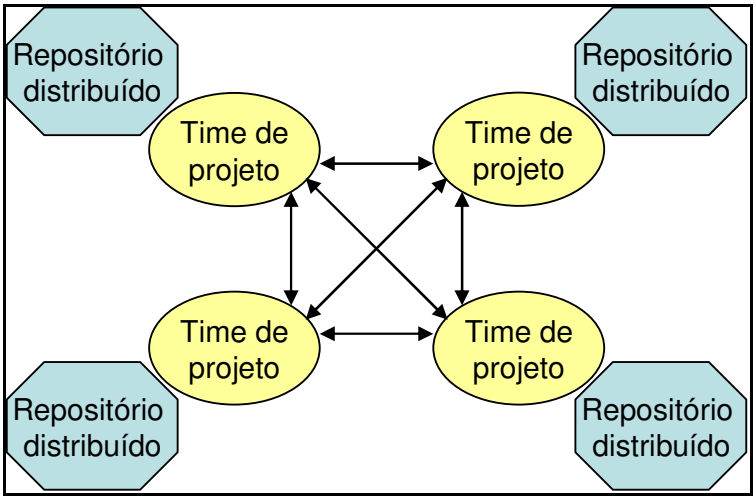


Figura 25 - Abordagem P2P (adaptado de [DES04]).

Este tipo de abordagem trás para a ER em DDS a vantagem de facilitar o compartilhamento e o controle do conhecimento de gerência de requisitos. Informações sobre requisições de alteração e sobre o acompanhamento do seu processo de avaliação, por exemplo, são muito dinâmicas, necessitando de constante atualização. Mas não há uma necessidade de transferência imediata deste conhecimento para outras equipes geograficamente dispersas, permitindo que este conhecimento possa ser armazenado localmente e disponibilizado sob demanda apenas para as equipes interessadas. Entretanto, esta abordagem pode dificultar a estruturação do conhecimento. Cada projeto e equipe podem utilizar seus próprios métodos para codificar e categorizar o conhecimento, dificultando assim a transferência deste conhecimento entre equipes geograficamente dispersas. Se cada equipe utilizar seus próprios métodos para armazenar as especificações de requisitos, por exemplo, com o passar do tempo esta documentação se tornará desestruturada.

Assim, esta abordagem torna-se adequada para a gestão do conhecimento em projeto. Proporciona a flexibilidade necessária para que cada equipe e projeto armazene o conhecimento dinâmico sobre a gerência de requisitos utilizando seus próprios processos e padrões. Mas não é a opção mais adequada para gestão do conhecimento sobre projetos e proveniente de projetos, pois pode levar a desestruturação deste tipo de informação.

## 6.4 Modelo de arquitetura de informação para gerência de requisitos em DDS

A abordagem híbrida apresenta características que a torna adequada ao contexto da ER em DDS. Para preservar os benefícios de cada abordagem e evitar suas limitações esta abordagem possui dois componentes. O primeiro é um repositório centralizado contendo o conhecimento sobre e proveniente de projetos. Este componente também mantém um índice para o segundo componente: conhecimento em projeto armazenado nos repositórios das equipes geograficamente distribuídas. A Figura 26 ilustra esta abordagem.

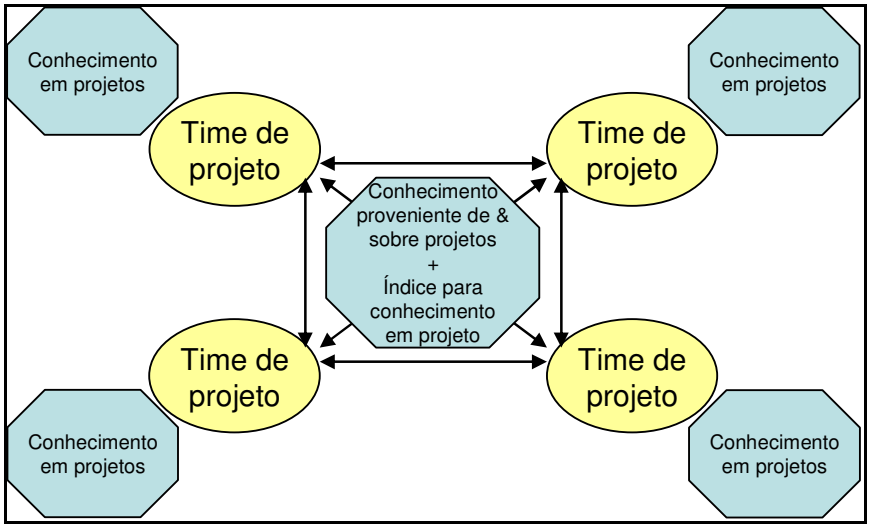


Figura 26 - Abordagem Híbrida (adaptado de [DES04]).

O uso de um repositório centralizado para o conhecimento sobre e proveniente de projetos propicia a padronização e estruturação do conhecimento de longo prazo sobre os requisitos das aplicações. Os padrões estabelecidos para a publicação da especificação de requisitos no repositório centralizado podem garantir que os requisitos sejam documentados de forma homogênea. Como exemplo de possíveis padrões pode-se citar o idioma exigido pela organização para os documentos publicados, a técnica de especificação de requisitos utilizada (casos de uso, linguagem natural, etc.), sistema métrico utilizado e glossário unificado de termos. Além disto, o uso de um repositório centralizado pode viabilizar a especificação dos requisitos com foco no produto de software, evitando que os requisitos sejam documentados para o escopo de cada projeto individual e que informações específicas de um determinado contexto possam prejudicar a interpretação sobre a especificação do produto de software em projetos futuros ou por outras equipes.

Um índice centralizado para o conhecimento em projeto, que por sua vez permanece armazenado nos repositórios distribuídos, oferece um mecanismo eficiente de coordenação e comunicação entre as equipes distribuídas. Por exemplo, uma equipe responsável pela execução de um projeto de manutenção em um determinado produto de software pode localizar a

especificação de requisitos que descreve o funcionamento do mesmo em um determinado momento. Mas pode também encontrar um índice para os demais projetos realizados sobre o mesmo produto de software, permitindo interagir diretamente com as equipes responsáveis pelos mesmos e obter informações mais detalhadas de seus repositórios sobre o conhecimento em projeto.

O conhecimento em projeto armazenado nos repositórios distribuídos, segundo componente da abordagem híbrida, proporciona um mecanismo eficiente e efetivo de captura do conhecimento em projeto. Uma vez que cada projeto é único e cada equipe é diferente, esta abordagem permite flexibilidade na adoção de processos de gerência de requisitos e padrões que sejam adequados à realidade de cada caso. Por exemplo, uma equipe em uma determinada localidade pode utilizar um sistema de automação de *workflow* para o acompanhamento das requisições de alteração de requisitos, enquanto outra pode simplesmente utilizar trocas de e-mails entre os *stakeholders* para a mesma finalidade. Também é possível evitar, com esta abordagem, a perda de informações que são relevantes apenas para um determinado projeto, mas que não agregam valor ao aprendizado organizacional ou não se enquadram nos padrões globais da organização. A priorização dos requisitos, por exemplo, é uma informação relevante durante a execução de um projeto, mas perde seu sentido ao termino do mesmo. Portanto, não há necessidade de sobrecarregar o repositório centralizado com informações que podem gerar resultados irrelevantes durante operações de busca por conhecimento.

### 6.4.1 Considerações sobre o modelo estático

As informações podem ser armazenadas nos repositórios conforme o modelo estático apresentado na Figura 27. O modelo utiliza a notação de diagrama de classes UML e representa, em um único modelo, tanto as informações mantidas no repositório centralizado quanto às informações mantidas no repositório distribuído. Esta estrutura é apresentada em um único modelo visando ilustrar mais claramente a integração entre os dois tipos de repositórios envolvidos.

Para representar o índice mantido no repositório centralizado, foi introduzido no diagrama o estereótipo *"<<índice>>"*. Todas as relações entre informações armazenadas em repositórios distintos foram representadas neste diagrama com este estereótipo. Tome-se como exemplo a relação intitulada "cria / altera" entre os elementos "Projeto" e "Produto". Esta relação permite determinar quais projetos foram realizados para a criação de um determinado produto de software e também quais projetos foram realizados visando à manutenção do mesmo produto de software. As informações sobre projetos são mantidas nos repositórios distribuídos, enquanto que as informações sobre os produtos de software são mantidas no repositório centralizado. Portanto, a relação entre estas informações será obtida através do índice.

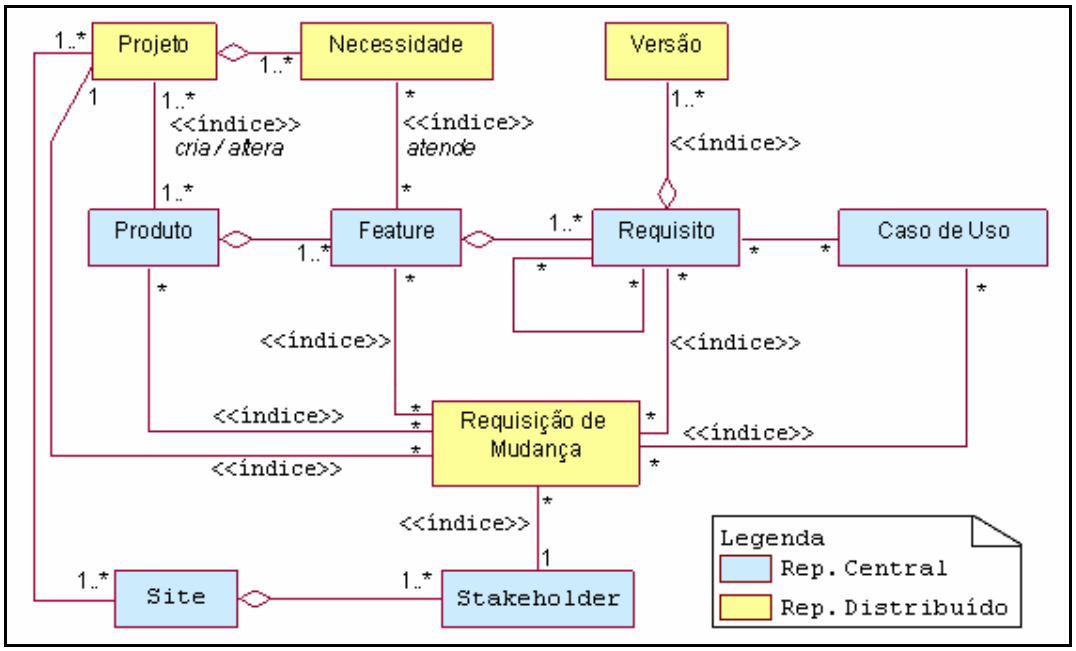


Figura 27 - Modelo estático dos repositórios.

Este modelo não esgota todas as possibilidades de registro de informações. Representa apenas uma sugestão de como o repositório pode ser implementado. Este modelo é um ponto de flexibilização do processo proposto. O objetivo com esta flexibilização é permitir que uma organização que deseje adotar o modelo possa adaptá-lo as características de seu processo de gerência de requisitos com maior facilidade, reduzindo assim o custo de implantação do mesmo. A entidade "Requisito", por exemplo, pode ser adaptada para o armazenamento de requisitos em linguagem natural, diagramas I* ou qualquer outro método, dependendo das necessidades e dos padrões utilizados pela organização. Nestes casos outros elementos poderiam ser incluídos neste modelo para suportar as particularidades destes métodos.

É importante também ressaltar que o modelo não prescreve como o repositório deve ser implementado. Limita-se apenas a descrever como as informações podem ser organizadas em cada tipo de repositório. Entretanto é fácil encontrar em várias tecnologias abstrações equivalentes às representadas no modelo. Por exemplo, case seja necessário implementar os repositórios em um SGBD, pode-se utilizar tabelas para implementar cada classe de informação apresentada e chaves estrangeiras para cada relacionamento entre estas classes. Caso seja necessário implementar os repositórios sob a forma de sites em uma Intranet corporativa, pode-se utilizar paginas *web* para implementar cada classe e *web links* para os relacionamentos entre as classes de informação.

### 6.4.2 Considerações sobre o modelo dinâmico

Do ponto de vista do modelo dinâmico, esta proposta não prescreve um conjunto

específico de atividades nem tão pouco exige uma estrutura rígida de papeis e responsabilidades para sua adoção. Novamente o objetivo é garantir a flexibilidade necessária para adoção desta proposta nas organizações. Entretanto, visando garantir os benefícios desejados, algumas características básicas devem estar presentes no processo de requisitos para adoção desta proposta. Estas características dizem respeito basicamente à necessidade de mudança de escopo dos processos, visando à elaboração de um processo focado no produto de software e na sua manutenção ao longo do tempo e dos diversos times envolvidos nos projetos.

Como foi visto na seção 2.3, [KOT98] apresenta uma abordagem de ER focada em projetos de desenvolvimento de software onde os *stakeholders* estão co-localizados. Esta abordagem pode ser utilizada em conjunto com a arquitetura de informação proposta na seção 6.4, mas necessita ser adaptada. O modelo de arquitetura de informação para gerência de requisitos em DDS aqui proposto, permite adaptar esta abordagem de forma a contemplar o ciclo de vida completo do software, o que inclui tanto o desenvolvimento do software quanto sua evolução ao longo de diversos projetos de manutenção. A Figura 28 ilustra o caráter integrado do modelo proposto e o contexto em que este se insere no processo de ER.

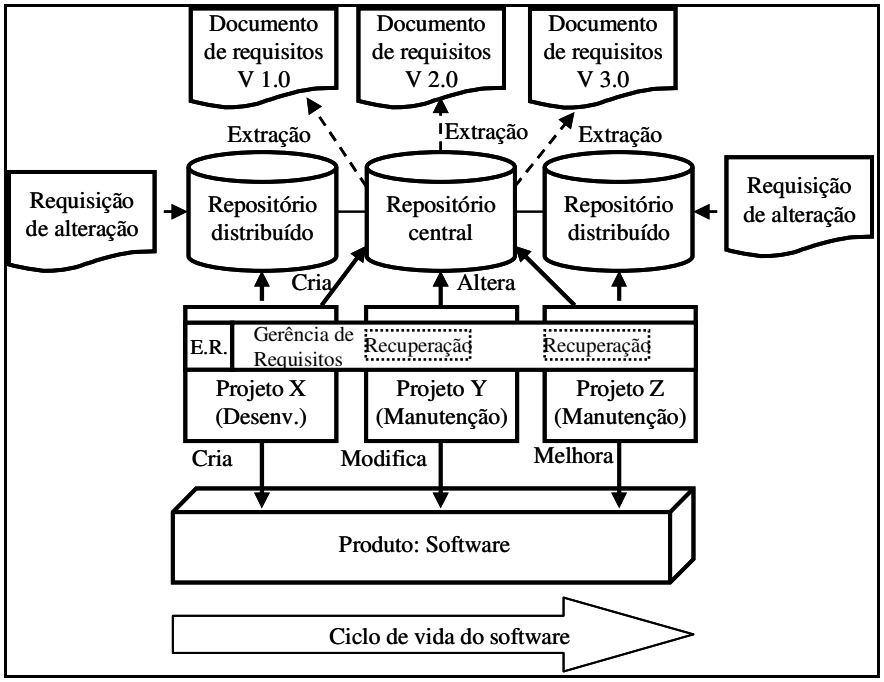


Figura 28 - Inserção do modelo no processo de ER.

Nesta abordagem, o processo de ER continua sendo utilizado no início do ciclo de vida do produto. A primeira diferença está no escopo de aplicação do processo. Neste caso ele não será usado no escopo exclusivo do projeto. Será utilizado no escopo do produto e levará em conta a necessidade de manutenção futura dos requisitos, levando assim a segunda diferença, que está no seu objetivo. Ao invés de simplesmente criar o documento de requisitos, este processo também tem o objetivo de criar um repositório central e o respectivo repositório distribuído para o

projeto. Assim, as informações podem ser armazenadas conforme proposto, deixando os repositórios prontos para futuras manutenções.

Após a criação dos repositórios de requisitos e aprovação da *baseline* inicial de requisitos, passa a ser executado o processo de gerência de requisitos. Este processo mantém a finalidade original, ou seja, gerenciar as alterações nos requisitos do software. A diferença é que nesta abordagem o processo não está restrito ao projeto de desenvolvimento do sistema, mas continua sendo executado ao longo de todo o ciclo de vida do software.

Em termos de papéis, esta proposta sugere a existência de um engenheiro de requisitos, tal como sugerido por [KOT98], com o acréscimo de responsabilidades extras. Além de executar atividades de elicitação, análise, negociação, documentação, validação e gerência de requisitos, o engenheiro de requisitos também será responsável por:

- Criação dos repositórios de informação;
- Manutenção dos repositórios de informação;
- Garantir a aderência à abordagem de gestão de requisitos aqui proposta.

Desta forma, caberão ao engenheiro de requisitos as seguintes atividades relacionadas a esta responsabilidade:

- Analisar as informações necessárias na organização. O conjunto total de informações pode varias dependendo da organização, do seu ramo de atividade, do domínio de aplicação para qual o produto é destinado e de necessidades específicas e inerentes ao negócio da organização;
- Categorizar a informação como conhecimento em projeto, conhecimento proveniente de projetos ou conhecimento sobre projetos. Esta caracterização é necessária para a aplicação da abordagem híbrida de gestão do conhecimento e é utilizada para determinar a estrutura dos repositórios;
- Projetar os repositórios. Com base nos passos anteriores é possível modelar a estrutura de dados necessária e alocar cada categoria de informação ao repositório mais adequado;
- Manter a estrutura dos repositórios. Ao longo do tempo e de acordo com necessidades futuras da organização e de seus projetos pode ser necessário adequar a estrutura dos repositórios incorporando novas informações ou novos relacionamentos entre as informações.

Com esta abordagem espera-se manter a especificação de requisitos atualizada e garantir uma fonte de informações que permita melhorar o conhecimento sobre o sistema legado, endereçando assim as duas principais dificuldades identificadas no estudo de caso 1.

# 7 Estudo de Caso 2

Neste capítulo são apresentados os resultados do estudo de caso 2. A seção 7.1 apresenta uma descrição do estudo de caso 2. A seção 7.2 apresenta a análise dos resultados da aplicação do instrumento de pesquisa. Por fim, a seção 7.3 apresenta as lições aprendidas neste estudo de caso.

## 7.1 Descrição

O estudo de caso 2 visava avaliar a aplicabilidade de um modelo de arquitetura de informação para gerência de requisitos em DDS e os benefícios decorrentes de sua utilização em projetos reais conduzidos em um ambiente DDS. Com este objetivo o modelo proposto foi implantado em um projeto de manutenção de software em um ambiente DDS na mesma organização onde foi realizado o estudo de caso 1.

Para implantação do modelo proposto foi utilizada a ferramenta *Rational RequisitePro*. Esta ferramenta foi escolhida por atender aos requisitos para implantação do modelo de arquitetura proposto. As Figuras 29 e 30 apresentam a estrutura final dos repositórios criados utilizando-se esta ferramenta. A Figura 29 apresenta a estrutura do repositório central e a Figura 30 apresenta a estrutura do primeiro repositório distribuído.

Após a implantação do modelo proposto e a execução do projeto, foi aplicado o instrumento de pesquisa. O instrumento de pesquisa adotado foi um questionário estruturado composto unicamente de questões em escala Likert de 5 níveis, variando de "Discordo Totalmente" (com peso 1) a "Concordo Totalmente" (com peso 5). Estas questões exploraram a percepção dos respondentes quanto a 5 dimensões. O protocolo de pesquisa encontra-se no Apêndice B deste volume.

A primeira dimensão do questionário, relativa aos dados demográficos, tem como objetivo identificar os respondentes quanto à formação acadêmica e profissional. Na segunda dimensão os respondentes foram convidados a expressar sua percepção quanto à contribuição do modelo para a redução das dificuldades para gerência de requisitos em DDS. Na terceira dimensão foi avaliada a percepção dos respondentes quanto à aplicabilidade do modelo. A quarta dimensão explora a percepção dos respondentes quanto à colaboração do modelo para a gestão do conhecimento sobre os requisitos da aplicação sendo mantida pelo projeto. Por fim, a dimensão 5 explora a percepção geral dos respondentes quanto aos benefícios e a aplicabilidade do modelo proposto.

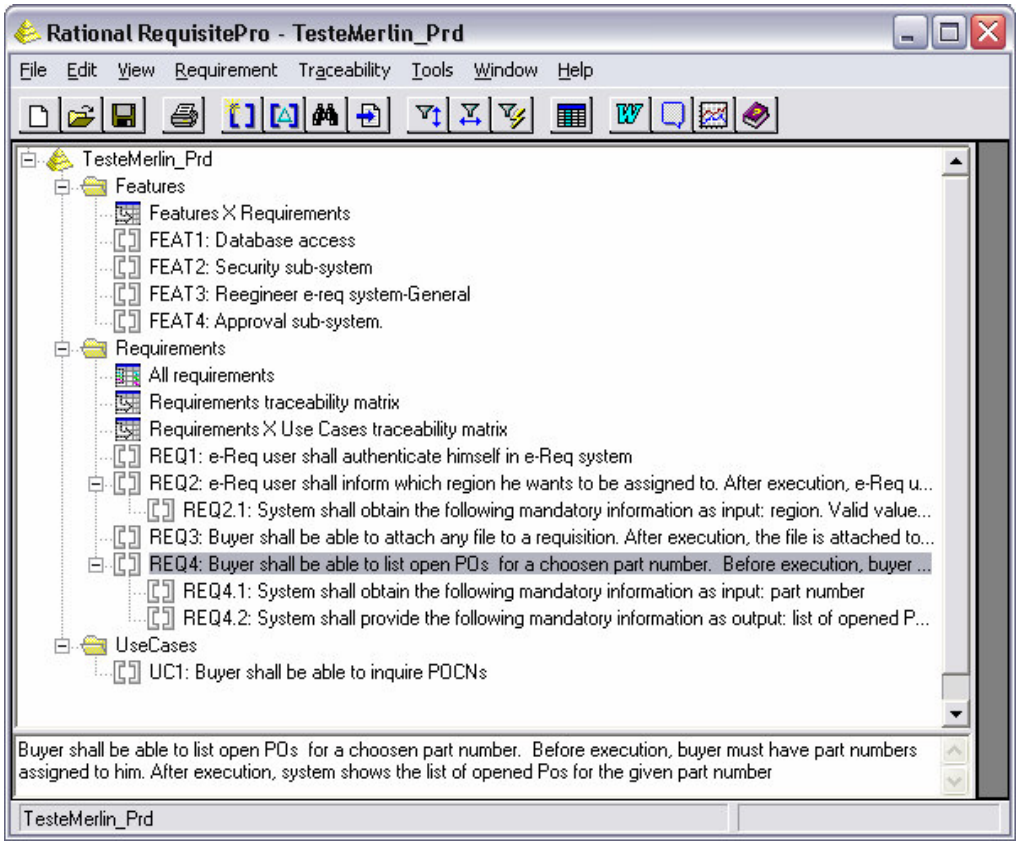


Figura 29 - Repositório central.

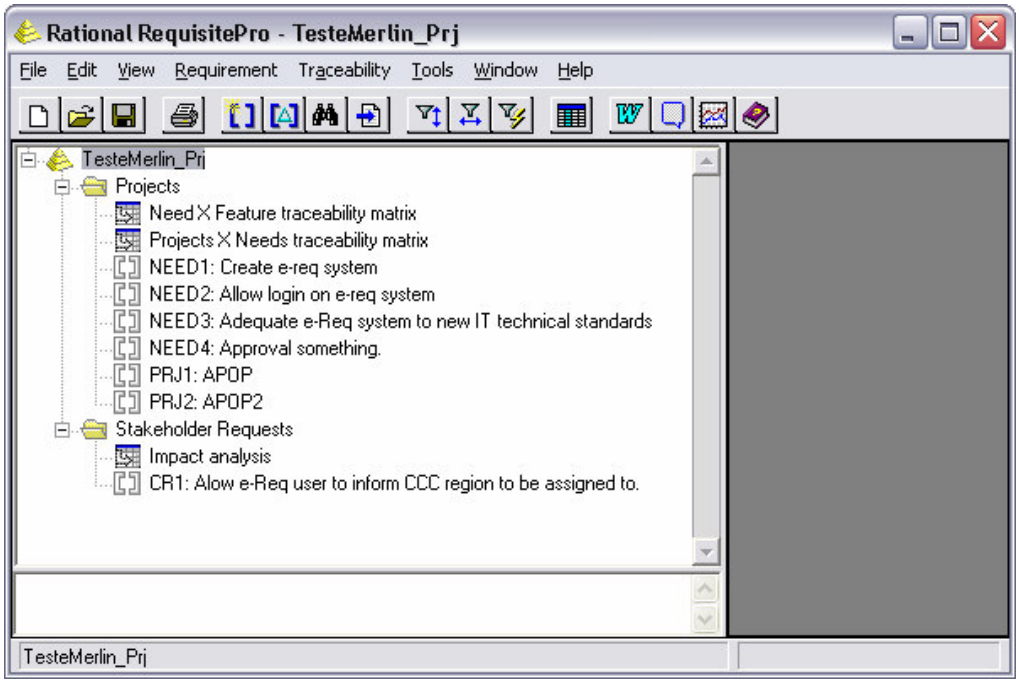


Figura 30 - Repositório distribuído.

## 7.2 Análise de dados

A análise dos resultados do estudo de caso foi realizada utilizando o módulo estatístico do Microsoft Excel. Os dados do questionário foram triangulados com documentação do projeto de forma a obter maior confiabilidade nos resultados. Foram também entrevistados alguns membros dos projetos para sanar dúvidas com relação às respostas obtidas no questionário.

A primeira dimensão do questionário, relativa aos dados demográficos, teve como objetivo identificar os respondentes, quanto à formação acadêmica e profissional. Foram entrevistados o gerente de projetos, o engenheiro de requisitos, o líder de desenvolvimento e o líder de testes. Todos os respondentes possuíam graduação em cursos relacionados à área de tecnologia da informação, sendo que um deles possuía mestrado em ciência da computação. A idade média dos respondentes foi de 28 anos e os mesmos possuíam em média 10 anos de experiência profissional em funções relacionadas à área de tecnologia da informação.

A segunda dimensão do questionário visava identificar a percepção dos respondentes quanto ao grau de contribuição do modelo proposto para a redução das dificuldades identificadas através do estudo de caso 1 e da pesquisa bibliográfica. A consolidação dos resultados é apresentada na Tabela 11.

Tabela 11 – Resultados da segunda dimensão do estudo de caso 2.

| Questão | Média |
|---|---|
| 8. O modelo permite reduzir as dificuldades relativas à ausência de documentação em projetos de manutenção. | 5,00 |
| 9. O modelo contribui para a preservação da documentação sobre requisitos ao longo de diversos projetos. | 4,50 |
| 10. O modelo facilita o acesso, em um ambiente DDS, às informações sobre os requisitos do software. | 4,25 |
| 11. O modelo facilita o gerenciamento da documentação sobre requisitos do software em ambientes DDS. | 4,25 |
| 12. O modelo contribui para a preservação do conhecimento sobre a aplicação. | 5,00 |

Os respondentes apresentaram unanimidade quanto à percepção sobre contribuição do modelo para a redução das dificuldades causadas pela ausência de documentação adequada em projetos de manutenção, questão 8. Desta forma, o modelo de arquitetura proposto apresenta um resultado satisfatório para redução da principal dificuldade para ER em projetos de manutenção de software em ambiente DDS, identificada tanto no estudo de caso 1 quanto na literatura.

Também houve unanimidade ao responderem que concordam totalmente que o modelo

contribui para a preservação do conhecimento sobre a aplicação, questão 12. Assim, o modelo de estrutura proposto apresenta um resultado satisfatório para redução da segunda maior dificuldade para ER em projetos de manutenção de software em ambiente DDS, identificada tanto no estudo de caso 1 quanto na literatura. Estas duas questões estão relacionadas e a os resultados, quando comparados, demonstram a coerência das respostas.

A questão 9 também alcançou um bom resultado, atingindo média de 4,5, demonstrando que, na percepção dos respondentes, o modelo contribui para a preservação da documentação sobre requisitos ao longo de diversos projetos. Entretanto não houve unanimidade entre os respondentes. A percepção dos respondentes demonstra que o modelo proposto contribui não apenas na redução causada pela ausência de documentação adequada, conforme demonstrado na questão 8, mas também contribui para a preservação da documentação para futuros projetos. Desta forma, o modelo proposto mostra-se adequado à mitigação de um problema previamente existente e à prevenção de sua propagação para projetos futuros.

Resultado satisfatório também foi observado com relação às questões 10 e 11, que exploravam a percepção dos respondentes quanto à contribuição do modelo em um ambiente DDS. Estas questões também demonstram que os respondentes concordam que o modelo facilita o acesso e o gerenciamento da documentação de requisitos em ambientes DDS, atingindo média de 4,25. De modo geral, todas as questões desta dimensão obtiveram médias satisfatórias, demonstrando assim uma boa percepção dos respondentes quanto à contribuição do modelo para redução das dificuldades listadas anteriormente.

A terceira dimensão do questionário visava avaliar a percepção dos respondentes quanto à aplicabilidade do modelo. A consolidação dos resultados é apresentada na Tabela 12.

Tabela 12 – Resultados da terceira dimensão do estudo de caso 2.

| **Questão** | **Média** |
|---|---|
| 13. O modelo é de fácil adoção. | 4,00 |
| 14. O modelo pode ser empregado independentemente da técnica de especificação de requisitos adotada. | 3,00 |
| 15. O modelo é independente das ferramentas utilizadas para sua implantação. | 3,75 |
| 16. O modelo torna o processo de gerência de requisitos significativamente mais caro. | 2,25 |
| 17. O modelo reduz significativamente a eficiência do processo de gerência de requisitos. | 1,00 |
| 18. O modelo contribui para o incremento da qualidade das especificações de requisitos. | 4,00 |
| 19. O modelo torna o processo de gerência de requisitos significativamente mais complexo. | 2,75 |

Os melhores resultados foram obtidos nas questões 13, 17 e 18 com médias 4, 1 e 4 respectivamente. Estes resultados demonstraram a percepção de que o modelo é de fácil adoção, não reduz significativamente a eficiência do processo de gerência de requisitos e contribui para o incremento na qualidade das especificações de requisitos. O resultado da questão 17 também reforça o resultado da questão 13, demonstrando coerência dos respondentes na avaliação do modelo proposto.

Dentre estas questões a que obteve maior variação entre os respondentes foi a questão 18, quanto a qualidade das especificações de requisitos. Este resultado demonstra que não há consenso entre os respondentes nesta questão. De fato, o modelo não trata diretamente a qualidade da especificação dos requisitos, já que está focado na gerência e não na elicitação dos requisitos. Para verificar este resultado, foi consultado o engenheiro de requisitos do projeto. Aparentemente, o bom resultado na média da resposta está relacionado com a melhoria significativa na estruturação das informações causada pela adoção do modelo proposto. De acordo com a percepção do engenheiro de requisitos do projeto, mesmo não tratando diretamente da elicitação de requisitos o modelo também contribui para o incremento da qualidade das especificações através de uma melhor estruturação das informações coletadas. Entretanto, a ausência de consenso entre os respondentes indica oportunidades de melhoria no modelo proposto.

Nas questões 14 e 15 a percepção dos respondentes apresentou-se neutra. A percepção dos respondentes não é convergente quanto à independência que o modelo apresenta da ferramenta utilizada para sua implantação. Estes resultados não eram esperados, tendo em vista que o modelo proposto foi elaborado também visando à independência da ferramenta utilizada na sua implantação. A percepção dos respondentes também não é convergente quanta à independência que o modelo apresenta da técnica de especificação de requisitos utilizada durante a elicitação de requisitos. Apesar de não esperados, estes resultados podem ser compreendidos como decorrência das limitações da pesquisa realizada. Os respondentes não possuíam parâmetros de comparação para avaliar esta afirmação, pois fizeram uma única aplicação utilizando uma única ferramenta e uma única técnica para especificação de requisitos. Para avaliar corretamente a independência do modelo quanto a estes fatores, serão necessários estudos futuros mais abrangentes utilizando, possivelmente, técnicas experimentais que permitam comprovar ou refutar a hipótese de independência do modelo e das ferramentas ou técnicas utilizadas em conjunto na sua adoção.

Também não há consenso quanto ao incremento de complexidade no processo de gerência de requisitos da organização, questão 19. A média mostrar-se ligeiramente favorável, mas com divergências entre os respondentes. Apesar de não apresentarem valores conclusivos, estes resultados não comprometem a aplicabilidade do modelo proposto. Entretanto, indicam

possibilidades de melhoria no modelo proposto.

Já a questão 16 a média de 2,75 demonstra uma leve tendência positiva na percepção dos entrevistados, que discordam da afirmação de que o modelo proposto encarece o processo de gerência de requisitos. Novamente, este resultado não é conclusivo, devido a divergência entre os entrevistados. De modo geral, as questões desta dimensão obtiveram médias positivas quanto à aplicabilidade do modelo proposto, sendo que nenhuma delas apresentou resultados desfavoráveis.

A quarta dimensão do questionário visava avaliar a percepção dos respondentes quanto à contribuição do modelo para a gestão do conhecimento sobre os requisitos da aplicação. A consolidação dos resultados é apresentada na Tabela 13.

Tabela 13 – Resultados da quarta dimensão do estudo de caso 2.

| **Questão** | **Média** |
|---|---|
| 20. O modelo facilita a transferência de conhecimento sobre requisitos entre equipes geograficamente dispersas. | 4,75 |
| 21. O modelo facilita o compartilhamento do conhecimento sobre requisitos entre equipes geograficamente dispersas. | 4,75 |
| 22. O modelo facilita o controle, por parte dos diversos times geograficamente distribuídos, do conhecimento sobre os requisitos. | 4,67 |
| 23. O modelo reduz a fragmentação da documentação sobre os requisitos do software. | 4,25 |
| 24. O modelo reduz a dispersão das informações sobre os requisitos do software. | 4,50 |
| 25. O modelo contribui para a padronização da documentação de requisitos. | 4,50 |

Os melhores resultados nesta dimensão foram obtidos nas questões referentes à gestão do conhecimento em DDS. As questões 20, 21 e 22 obtiveram médias altas, demonstrando consenso entre os respondentes na percepção de que o modelo contribui para a transferência, o compartilhamento e o controle, por parte dos diversos times geograficamente distribuídos, do conhecimento sobre os requisitos. Resultados igualmente satisfatórios foram atingidos nas questões 23, 24 e 25, referentes à redução da fragmentação, da dispersão e a padronização da documentação de requisitos. De modo geral, a maioria das questões desta dimensão obteve médias altas, demonstrando assim que a percepção dos respondentes é satisfatória quanto à contribuição do modelo para a gestão do conhecimento sobre os requisitos.

A quinta dimensão do questionário visava avaliar a percepção geral dos respondentes quanto à contribuição do modelo proposto. A consolidação dos resultados é apresentada na Tabela 14.

Tabela 14 – Resultados da quinta dimensão do estudo de caso 2.

| Questão | Média |
|---|---|
| 26.Em termos gerais, o uso do modelo foi benéfico para o projeto. | 5,00 |
| 27. Em termos gerais, foi mais fácil gerenciar os requisitos com o uso do modelo. | 4,25 |
| 28. O modelo pode ser aplicado tanto em projetos de desenvolvimento quanto em projetos de manutenção de software. | 4,25 |

A questão 26 demonstra a unanimidade dos respondentes na percepção de que o modelo proposto foi benéfico para o projeto. Esta percepção é coerente com os resultados encontrados nas dimensões 2 e 4. A questão 27, apresentou a percepção de que, em termos gerais, foi mais fácil gerenciar os requisitos com o uso do modelo proposto, confirmando assim a aplicabilidade do modelo. Por fim, a questão 28 demonstra a percepção de que o modelo pode ser aplicado tanto em projetos de manutenção quanto de desenvolvimento de software. Este resultado é importante, por reforçar a generalização do modelo proposto, conforme descrito no capítulo 7.

## 7.3 Lições aprendidas

Com base nos resultados da analise de dados e nas observações realizadas foram identificadas as lições aprendidas do estudo de caso 2. Estas lições servirão como base para futuras melhorias no modelo de arquitetura de informação para gerência de requisitos em DDS. Na seqüência são apresentadas as lições aprendidas:

**Lição 1: Gerenciar os requisitos com foco no produto de software contribui para a preservação do conhecimento sobre a aplicação.**

Durante a pesquisa bibliográfica e a realização do primeiro estudo de caso observou-se uma tendência para a aplicação da ER com foco em projetos e não nos produtos criados e mantidos pelos mesmos. Esta abordagem causava a fragmentação da documentação relativa aos requisitos de software e também favorecia a perda do conhecimento sobre as aplicações ao longo de diversos projetos. De acordo com a percepção dos entrevistados a adoção de um modelo que possibilita o gerenciamento dos requisitos de forma desatrelada dos projetos torna possível evitar a fragmentação da documentação e a preservação do conhecimento sobre os requisitos do software. Esta lição também demonstra que a relevância dos requisitos do software transcende os curtos períodos de duração dos projetos, alinhando-se assim com autores como [LIU99] e [ZAN03] que defendem a importância da compreensão dos requisitos do sistema legado para projetos de manutenção de software.

**Lição 2: A mudança de paradigma necessária para a troca de uma abordagem de engenharia de requisitos focado em projetos para uma abordagem focado em**

**produto não é uma tarefa trivial.**

Durante a fase de treinamento da equipe de projeto foi necessário um esforço imprevisto para que a equipe assimilasse a diferença entre realizar a ER com foco no produto ao invés de realizá-la com foco no projeto. Para alguns indivíduos não havia distinção entre o projeto e o produto em que trabalham. Também não havia distinção entre a descrição dos requisitos (atividade típica de análise) e a descrição das soluções (atividade típica de projeto). Assim, estes indivíduos tinham uma forte inclinação a registrar a descrição da solução que seria desenvolvida enquanto julgavam estar descrevendo requisitos de software. Portanto, esta mudança de paradigma parece não ser trivial.

Esta dificuldade pode estar relacionada com as praticas de orientação a projeto que são amplamente adotadas no mercado de software, tais como as sugeridas por modelos como o CMM e pelo PMI. Estas práticas, quando mal interpretadas ou mal adotadas, podem provocar pressões por resultados e metas vinculadas unicamente aos projetos, levando os profissionais a não observarem as necessidades de longo prazo que não apresentam impacto imediato para seus projetos. A documentação dos sistemas é um exemplo deste comportamento. Quando um projeto inicia sem uma documentação adequada, as equipes de projeto consideram o fato como um risco alto para seus projetos. Mas por outro lado, as mesmas equipes consideram o esforço necessário para documentar seus produtos como um custo que pode ser evitado sem prejuízo ao projeto, ignorando os prejuízos de longo prazo causados com esta decisão.

**Lição 3: Reconhecer a singularidade de cada projeto contribui para gestão do conhecimento sobre requisitos em ambientes DDS.**

O reconhecimento de que cada projeto é único e que cada equipe de projeto possui diferentes necessidades é fundamental para viabilizar um modelo de arquitetura de informação para gerência de requisitos em DDS. A centralização de todas as informações poderia ser o caminho ideal após a identificação das dificuldades causadas pela fragmentação e pela dispersão da documentação sobre requisitos. Entretanto, esta centralização não seria a melhor solução para projetos distribuídos, pois careceria da flexibilidade necessária para acomodar as diferenças entre cada projeto e cada equipe, dificultando assim o trabalho de equipes geograficamente dispersas. O modelo híbrido adotado permitiu tratar o problema da fragmentação, dispersão e perda de informações ao longo do tempo sem comprometer a eficiência de cada projeto envolvido na criação ou manutenção de um determinado produto de software.

**Lição 4: O modelo proposto necessita de melhorias visando tornar mais claros os pontos de flexibilidade que o tornam independente das técnicas de especificação de requisitos utilizadas.**

Neste estudo de caso a percepção dos respondentes demonstrou-se neutra no que se refere à independência existente entre o modelo proposto e a técnica de especificação de

requisitos adotada. Apesar da relação entre este resultado não conclusivo e as limitações do instrumento de pesquisa, melhorias também poderiam ser introduzidas no modelo visando tornar mais claros os pontos de flexibilidade que permitem que o modelo seja adotado com o uso de outras técnicas de especificação de requisitos.

**Lição 5: Os requisitos necessários para determinar a compatibilidade entre uma determinada ferramenta e o modelo proposto precisam ser claramente definidos.**

A percepção dos respondentes demonstrou-se favorável a afirmação de que o modelo é independente das ferramentas utilizadas para sua implantação. Mas o alto desvio padrão encontrado demonstra que não há consenso entre os respondentes nesta questão. Novamente, este resultado está relacionado às limitações do instrumento de pesquisa. Mesmo assim, este resultado demonstra a necessidade de melhorias visando tornar claros os requisitos para que uma determinada ferramenta possa ser utilizada para implantação do modelo proposto.

**Lição 6: O estudo de caso 2 apresentou bons resultados quanto à aplicabilidade do modelo proposto.**

Com base na percepção dos respondentes foi possível verificar a aplicabilidade do modelo proposto. Em termos gerais o modelo demonstrou-se de fácil utilização e benéfico para o projeto. Alguns pontos de melhoria foram identificados, mas não foram encontrados fatores que comprometessem a adoção do modelo ou que pudessem trazer riscos ao projeto após sua adoção. O modelo também apresentou resultados satisfatórios no tratamento das dificuldades para gerência de requisitos em projetos de manutenção de software em DDS. Confirmando assim a viabilidade de sua adoção.

Outro resultado positivo e favorável a adoção do modelo proposto foi a percepção dos respondentes de que o modelo pode ser aplicado tanto em projetos de desenvolvimento quanto em projetos de manutenção de software. Confirmando assim a hipótese, desenvolvida ao longo desta pesquisa, de que as dificuldades identificadas durante a pesquisa são apenas evidenciadas durante os projetos de manutenção de software e não causadas pelos mesmos.

# 8 Conclusão

Os sistemas de Informação são peças fundamentais para o funcionamento das organizações modernas. É natural que a área de tecnologia da informação torne-se peça chave para viabilizar o sucesso das organizações. Dada a importância do software no mundo moderno, o alto investimento realizado no seu desenvolvimento, a natureza dinâmica dos negócios e a crescente competitividade observada na indústria, a manutenção de software torna-se inevitável. Atualmente, estima-se que a maior parte de todo o investimento realizado pela indústria em engenharia de software seja aplicada em projetos de manutenção de software.

Por outro lado, a globalização tem levado as organizações a distribuírem ao redor do mundo seus esforços em engenharia de software. Este movimento busca a redução de custos através da contratação de mão de obra especializada em paises como Brasil e Índia, dentre outros. Estes países oferecem atualmente mão de obra mais barata, incentivos fiscais para o investimento em tecnologia e uma sinergia cultural que favorece o trabalho intelectual.

Os temas abordados nesta dissertação são, portanto, considerados de relevância significativa para a pesquisa em engenharia de software e vem despertando crescente interesse tanto no meio acadêmico quanto na indústria de software. Pesquisadores e engenheiros de software buscam desenvolver soluções para a área da ER em DDS. Entretanto, não foram encontrados estudos significativos sobre o tema no campo da manutenção de software. Esta pesquisa buscou explorar estes temas identificando os desafios que emergem e propondo soluções.

Entretanto, o modelo proposto não é restrito à manutenção de software. Com a evolução da pesquisa percebeu-se que o modelo seria benéfico também para projetos de desenvolvimento, sendo sua implantação inclusive mais barata e eficiente quando realizada nesta etapa inicial do ciclo de vida do software. Assumiu-se nesta pesquisa a hipótese de que as dificuldades existentes são apenas evidenciadas durante os projetos de manutenção de software e não causadas pelos mesmos. Os estudos realizados ainda são limitados para confirmação definitiva desta hipótese. Mas os resultados do estudo de caso 2 representam indícios relevantes rumo à confirmação da mesma. Os respondentes demonstraram a percepção de que o modelo pode ser empregado tanto em projetos de manutenção de software quanto em projetos de desenvolvimento de software.

Conforme apresentado no capítulo 7, o objetivo geral desta pesquisa foi atingido. Foi apresentado nesta dissertação de mestrado um modelo de arquitetura de informação para gerência de requisitos em DDS. Os objetivos específicos também foram atingidos. O referencial

teórico apresentado no capítulo 2 é fruto do esforço em aprofundar o conhecimento em gerência de requisitos, manutenção de software, DDS e tópicos relacionados. O estudo de caso 1 permitiu identificar o processo de ER seguido por uma organização para manutenção de software em um ambiente DDS e as principais dificuldades enfrentadas neste contexto. Também foi elaborada uma proposta preliminar de modelo de arquitetura de informação para gerência de requisitos em DDS. Por fim, e sua aplicabilidade e os benefícios de sua utilização foram avaliados através do estudo de caso 2.

Os resultados do estudo de caso 2 demonstraram a aplicabilidade do modelo. Em termos gerais o modelo demonstrou-se de fácil utilização e benéfico para o projeto. Entretanto, duas possibilidades de melhoria foram identificadas. A primeira diz respeito à independência entre o modelo e as técnicas de especificação de requisitos. O modelo proposto necessita de melhorias visando tornar mais claros os pontos de flexibilidade que o tornam independente das técnicas de especificação de requisitos utilizadas. A segunda oportunidade de melhoria diz respeito aos requisitos que uma ferramenta deve apresentar para viabilizar sua utilização na implantação do modelo proposto. Os requisitos necessários para determinar a compatibilidade entre uma determinada ferramenta e o modelo proposto precisam ser claramente definidos.

## 8.1 Contribuições

A proposta de um modelo de arquitetura de informação para gerência de requisitos em DDS visa contribuir para a área de engenharia de software ao preencher uma lacuna existente na área de DDS, especificamente no que se refere à ER e à manutenção de software. Os estudos realizados durante esta pesquisa indicam a necessidade de desenvolvimento de novos processos e técnicas que permitam lidar com as dificuldades presentes neste novo contexto. Esta necessidade também é corroborada por outros estudos, tais como [DAM02] e [ZOW02], que sugerem a necessidade de que pesquisas de campo devem ser realizadas para que entenda-se o impacto dos problemas de comunicação e coordenação nas atividades de ER, identifique-se os desafios apresentados pelas organizações distribuídas e apresente-se recomendações para superar os problemas associados com a distância.

Além disso, este estudo apresenta novos dados empíricos e busca contribuir também para a pesquisa na área de engenharia de software através do emprego de métodos qualitativos de pesquisa científica. Pode-se justificar o uso de métodos qualitativos, tais como o estudo de caso, aqui empregado, pelo fato desta pesquisa envolver o estudo da ER e da manutenção de software em ambientes DDS no seu contexto real, com a descrição e a compreensão do estado da arte naquelas situações em que a prática se antecipa à teoria, conforme sugerido por [HOP97]. Os dois estudos de caso utilizados nesta pesquisa foram realizados em um centro de

desenvolvimento de software de uma multinacional de grande porte que utiliza intensivamente a prática de DDS.

Finalmente, este estudo visa contribuir com a prática ao atender uma demanda organizacional crescente por melhorias nos processos de ER, bem como, por tratar das dificuldades causadas pela distribuição das equipes de desenvolvimento. O modelo proposto já esta em uso na organização que participou dos dois estudos de caso e outros tópicos para pesquisas futuras continuam sendo identificados.

## 8.2 Limitações do estudo

A principal limitação deste estudo está associada à estratégia adotada na pesquisa empírica realizada. A generalização dos resultados fica restrita em virtude do pequeno número de projetos e respondentes envolvidos nos estudos de caso realizados. Entretanto, o estudo busca embasamento em uma forte pesquisa na base teórica, como é típico de estudos de natureza exploratória e base qualitativa, permitindo assim uma boa segurança nas conclusões apresentadas.

Outra limitação importante diz respeito ao período de duração desta pesquisa. A manutenção de software é um esforço de longo prazo que transcende o escopo de projetos individuais. Seria necessária uma pesquisa igualmente de longo prazo para avaliar mais profundamente os efeitos da aplicação de modelo de arquitetura de informação para gerência de requisitos em DDS. Infelizmente, não foi possível realizar tal pesquisa dada à limitação imposta pelo cronograma deste trabalho.

## 8.3 Trabalhos futuros

Identificou-se ao longo desta pesquisa potencial para realização de trabalhos futuros no tema de gerência de requisitos em DDS. Como pesquisas futuras sugerem-se:

- Realização de pesquisa de longo prazo através de todo o ciclo de vida de um produto de software, utilizando-se possivelmente de técnicas experimentais e quantitativas, para o acompanhamento da evolução da especificação de requisitos. Este estudo teria como finalidade explorar os efeitos da dispersão temporal no modelo proposto.
- Realização de pesquisa envolvendo mais organizações de diferentes localidades para explorar mais profundamente os efeitos da dispersão geográfica no modelo proposto.

# REFERÊNCIAS


[BER04] BERRY, D. M.; DAUDJEE, K.; DONG, J.; FAINCHTEIN, I.; NELSON, M. A.; NELSON, T.; OU, L. "User's manual as a requirements specification: case studies". Requirements Engineering Journal, vol. 9, fevereiro 2004, pp. 67-82.

[BIA03] BIANCHI, A.; CAIVANO, D.; MARENGO, V.; VISAGGIO, G. "Iterative Reengineering of Legacy Systems". IEEE Transactions on Software Engineering. vol. 29, no. 3, March 2003. pp. 225-241.

[BRO95] BRODIE, M. L.; STONEBRAKER, M. "Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach". San Francisco: Morgan Kaufmann Publishers, Inc. 1995, 210p.

[CAN72] CANNING, R. "The Maintenance 'Iceberg'". EDP Analyzer. vol. 10, no. 10, October 1972.

[CAR99] CARMEL, E. "Global Software Teams – Collaborating Across Borders and Time Zones". Prentice Hall. 1999. 269p.

[DAM02] DAMIAN, D.; ZOWGHI, D. "The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization". In: IEEE Joint International Conference on Requirements Engineering (RE'02), 2002, Essen, Germany. Proceedings… IEEE Computer Society, 2002, p. 319-328.

[DES04] DESOUZA, K. C.; EVARISTO, J.R. "Managing Knowledge in Distributed Projects". Comunications of the ACM. Abril 2004, Vol. 47, No.4. pp 87-91.

[DMM02] DAMM, D.; SCHINDLER, M. "Security issues of a knowledge médium for distributed projects work". Intern. J. of Project Management, Vol. 20, 2002. pp 37-44.

[EBN02] EBNER, G.; KAINDL, H. "Tracing All Around in Reengineering". IEEE Software, May 2002, pp.70-76.

[ESP04] ESPINDOLA, R. S.; MAJDENBAUM, A.; AUDY, J. L. N. "Uma Análise Crítica dos Desafios para Engenharia de Requisitos em Manutenção de Software". In: VII Workshop on Requirements Engineering, 2004, Tandil, Argentina, 2004.

[ESP05] ESPINDOLA, R. S.; MAJDENBAUM, A.; AUDY, J. L. N. "Requirements Engineering Challenges for Software Maintenance Projects in Distributed Software Development Environments". 21st International Conference on Software Maintenance. 2005.

[ESP05b] ESPINDOLA, R. S.; LOPES, L. T.; PRIKLADNICKI, R.; AUDY, J. L. N. "Uma Abordagem Baseada em Gestão do Conhecimento para Gerência de Requisitos em Desenvolvimento Distribuído de Software". In: VIII Workshop on Requirements Engineering, 2005, Porto, Portugal, 2005.

[EVA00] EVARISTO, R.; SCUDDER, R. "Geographically distributed project teams: a

dimensional analysis". In: HICSS, 2000, Havaí. Proceedings… EUA, p. 1-15, 2000.

[EVA05] EVARISTO, R.; AUDY, J. L. N.; PRIKLADNICKI, R.; AVRITCHIR, J. "Wholly Owned Offshore Subsidiaries for IT Development: A Program of Research". In: Proc. of the 38th Hawaii International Conference on System Sciences - HICSS. Hawaii, USA, 2005.

[FRA98] FRANCH, X.; BOTELLA, P. "Putting Non-Functional Requirements into Software Architecture". 9th International Workshop on Software Specification. IEEE Software. 1998.

[FUG00] FUGGETTA, A. "Software Process: A Roadmap". Future of Software Engineering. ACM. 2000. pp 25-34.

[GIL95] GIL, A. "Métodos e Técnicas de pesquisa social". São Paulo: Atlas, 1995.

[GOG96] GOGUEN, J. "Formality and Informality in Requirements Engineering". Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96). IEEE. 1996.

[HAN93] HANNA, M. "Maintenance Burden Begging for Remedy". Software Magazine, April 1993, pp. 53-63.

[HAN99] HANSEN, M.T.; NOHIRA, N.; TIERNEY, T. "What's your strategy for managing knowledge?". Harvard Business Review 77, 2 (Mar./Apr. 1999), pp 106-116.

[HER01] HERBSLEB, J.; MOITRA, D. "Global Software Development". IEEE Software. 2001, 5p.

[HOP97] HOPPEN, N. "Avaliação de artigos de pesquisa em sistemas de informação: proposta de um guia". In XXI Congresso da ANPAD, Rio de Janeiro. Anais. Brasil, 1997.

[HUM89] HUMPHREY, W.; KITSON, D.; KASSE, T. "The State of Software Engineering Practice: A Preliminary Report". ACM. 1989.

[IEE98] IEEE, Institute. "IEEE Std 1219-1998. IEEE Standard for Software Maintenance". New York: Institute of Electrical and Eletronic Engineers. Inc., 1998, 52p.

[INC04] International Council on Systems Engineering. "What is Systems Engineering?". Capturado em: http://www.incose.org/practice/whatissystemseng.aspx , Agosto 2004.

[ISO98] ISO. "Information Technology – Guide for ISO/IEC 12207 (Software Life Cycle Processes)". Genebra: International Organization for Standardization, 1998, 49p.

[ISO99] ISO. "ISO/IEC 14764 – Software Maintenance". Genebra: International Organization for Standardization, 1999, 38p.

[JAC01] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. "The Unified Software Development Process". Upper Saddle River, NJ: Addison-Wesley, 2001.

[KAR98] KAROLAK, D. "Global Software Development – Managing Virtual Teams and Environments". IEEE Computer Society. Los Alamitos, EUA. 1998. 159p.

[KIE03] KIEL, L. "Experiences in Distributed Development: A Case Study", In. Workshop on Global Software Development at ICSE, Oregon, EUA. 2003.

[KOT98] KOTONYA, G.; SOMMERVILLE, I. "Requirements Engineering: process and techniques". New York: John Wiley & Sons Ltd, 1998, 282p.

[KRU00] KRUCHTEN, P. "The Rational Unified Process: An Introduction". Upper Saddle River, NJ: Addison-Wesley, 2000.

[LEF03] LEFFINGWELL, D.; WIDRIG, D. "Managing Software Requirements – A Use Case Approach". Addison-Wesley. 2003, 492p.

[LIU99] LIU, K.; ALDERSON, A.; QURESHI, Z. "Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour". Proceedings of the International Conference on Software Maintenance, 1999, IEEE Computer Society, Los Alamitos, pp3-12.

[LOP03] LOPES, L. T.; AUDY, J. L. N. "Em busca de um modelo de referência para engenharia de requisitos em ambientes de desenvolvimento distribuído de software". Proceedings of 6th International Workshop on Requirements Engineering, Piracicaba, Brasil, 2003. p. 78-92.

[LOP03b] LOPES, L. T.; AUDY, J. L. N.; PRICKLADNICKI, R.; MAJDENBAUM, A. "Uma proposta para processo de requisitos em ambientes de desenvolvimento distribuído de software". Proceedings of 6th International Workshop on Requirements Engineering, Piracicaba. Brasil, 2003. p. 329-342.

[LOP04] LOPES, L. T.; AUDY, J. L. N,; MAJDENBAUM, A. "Distributed Requirements Specification: Minimizing The Effect Of Geographic Dispersion". Proceedings of ICEIS 2004.

[LUC01] LUCIA, A.; FASOLINO, A. R.; POMPELLA, E. "A Decisional Framework for Legacy System Management". In: International Conference on Software Maintenance, 2001, 10p.

[MAR83] MARTIN, R.J.; OSBORNE, W. "Guidance of Software Maintenance". U.S. Nat. Bureau of Standards. NBS 500-129, 1983.

[MIC04] Microsoft Corporation. "Microsoft Solutions Framework Version 3.0 Overview". Capturado em: http://www.microsoft.com/msf/ , Maio 2004.

[NUS00] NUSEIBEH, B.; EASTERBROOK, S. "Requirements Engineering: A Roadmap". ACM - Future of Software Engineering. 2000. pp 37-45.

[PRE01] PRESSMAN, R. S. "Software Engineering: a practitioner's approach". New York: McGraw Hill, 5th ed., 860p.

[PRE95] PRESSMAN, R. S. "Engenharia de Software". São Paulo: Makron Books, 1995, 1056p.

[PRI03] PRICKLADNICKI, R.; AUDY, J. L. N.; EVARISTO, R. "Requirements Management in Global Software Development: Preliminary Findings from a Case Study in a SW-CMM context". In: International Workshop on Global Software Development at

ICSE, 2003, Oregon. Proceedings… EUA. May. 2003.

[PRI04] PRIKLADNICKI, R.; AUDY, J. L. N. "MuNDDoS: Um Modelo de Referência para Desenvolvimento Distribuído de Software". In: XVIII SBES - SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, Brasília. 2004.

[RAT96] RATIONAL Software Corporation. "A Rational Development Process. White Paper". Capturado em http://www.rational.com/products/rup/whitepapers.jsp , Setembro 2002.

[RAT98] RATIONAL Software Corporation. "Rational Unified Process: Best Practices for Software Development Teams". White Paper. Capturado em http://www.rational.com/products/rup/whitepapers.jsp , Agosto 2002.

[REI00] REIFER, D. J. "Requirements Management: The Search for Nirvana". IEEE Sofware. 2000.

[ROC01] ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. "Qualidade de Software: Teoria e Prática". São Paulo: Prentice Hall, 2001, 303p.

[SAY05] SAYÃO, M.; Leite, J. C. S. P. "Uso de Agentes no Processo de Requisitos em Ambientes Distribuídos de Desenvolvimento" Anais do WER05 - Workshop em Engenharia de Requisitos, Porto, Portugal, Junho 13-14, 2005, pp 135-147.

[SID96] SIDDIQI, J.; SHEKARAN, M. C. "Requirements Engineering: the emerging wisdom". IEEE Software. 1996.

[SNE03] SNEED, H.M. "Critical Success Factors in Software Maintenance". In: International Conference on Software Maintenance, 2003, 9p.

[SOM03] SOMMERVILLE, I. "Engenharia de Software". 6a edição. Addison Wesley. 2003.

[SOM97] SOMMERVILLE, I.; SAWYER, P. "Requirements Engineering – a good practice guide". New York: John Wiley & Sons Ltd, 1997, 391p.

[SOU98] SOUZA, M. J. C. "A Survey on the Software Maintenance Process". In: International Conference on Software Maintenance, 1998, 10p.

[STE98] STEVENS, P.; POOLEY, R. "Systems Reengineering Patterns". In: Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering, vol. 23 issue 6, 1998, pp. 17-23.

[THA90] THAYER, R.; DORFMAN, M. "System and Software Requirements Engineering". IEEE Computer Society Press Tutorial. 1990. 718p

[UMA97] UMAR, A. "Application (Re)Engineering: Building Web-Based Applications and Dealing with Legacies". Upper Saddle River: Prentice Hall PTR, 587p.

[WHI95] WHITE, S. M. "Capturing Requirements for Legacy Systems". Proceedings of the International Symposium and Workshop on Systems Engineering of Computer Based Systems, 1995. pp 251-256.

[YIN01] YIN, R. "Estudo de Caso: planejamento e métodos". São Paulo: Bookman, 2001, 205p.

[ZAN03] ZANLORENCI, E. P.; BURNETT, R. C. "Abordagem de Engenharia de Requisitos em Software Legado". Anais do WER03 - Workshop em Engenharia de Requisitos, Piracicaba-SP, Brasil, 2003, pp 270-284.

[ZOW02] ZOWGHI, D. "Does Global Software Development need a diferent requirements engineering process?". Proceedings of International Workshop on Global Software Development 2002. 2002. 3p.

# APÊNDICE A – Protocolo de pesquisa do estudo de caso 1

## Engenharia de requisitos em Projetos de Manutenção de Software

**Objetivo**
Identificar o processo de engenharia de requisitos seguido pela organização em projetos de manutenção de software, visando avaliar o nível de aderência ao padrão definido no *Quality Framework* da organização e os principais problemas enfrentados neste contexto.

**Questão de pesquisa**
Como é realizada a engenharia de requisitos em projetos de manutenção?

**Unidade de estudo**
Projetos de manutenção de software da organização.

**Característica-chave do método de estudo de caso**
Este é um roteiro para o desenvolvimento e aplicação de um instrumento de pesquisa semi-estruturada com questões abertas que se caracteriza como uma pesquisa do tipo transeccional.

**Organização deste protocolo**
Este protocolo está organizado como segue:

1. **Procedimentos**

| A. Reuniões para levantamento das questões e estruturação do roteiro de entrevista | |
|---|---|
| Participantes: | Rodrigo Santos de Espindola<br>Prof. Dr. Jorge Luis Nicolas Audy. Doutor em Sistemas de Informação – UFRGS - 2001 (Especialista, Pesquisador Sênior) |
| Local: | AGT – Agencia de Gestão Tecnológica |
| Data: | 09/07/2004 |

| B. Validação de face e conteúdo | |
|---|---|
| Participantes: | Prof. Dr. Marcelo Blois. Doutor em Ciência da Computação – PUC-RJ – 2002 (Especialista, Pesquisador Sênior)<br>Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 12/07/2004 |

| C. Adequação do roteiro de entrevista com base na validação de face e conteúdo | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 12/07/2004 |

| D. Pré-teste | |
|---|---|
| Participantes: | Rodrigo Santos de Espindola<br>Elson Zago – Tech Leader |
| Local: | Dell GDC – Global Development Center |
| Data: | 16/07/2004 |
| Participantes: | Rodrigo Santos de Espindola<br>Azriel Majdenbaum – System Architect |
| Local: | Dell GDC – Global Development Center |
| Data: | 19/07/2004 |

| E. Adequação do roteiro de entrevista com base no pré-teste | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 20/07/2004 |

| F. Autorização da empresa participante |
|---|

| | |
|---|---|
| Participantes: | Ana Trindade (Dell Computadores do Brasil) |
| Local: | Autorizado por e-mail |
| Data: | 20/07/2004 |

| G. Formato de aplicação do instrumento | | | |
|---|---|---|---|
| Tipo: | Entrevista semi-estruturada com questões abertas | | |
| Entrevistador: | Rodrigo Santos de Espindola | | |
| Caso 1 | SMS - Loja On-line | | |
| Entrevistado | Nome | Data | Local – Sala |
| Delivery Manager | Luiz Souto | 29/07/04 | GDC – Limerick |
| Project Manager | Marco Rigo | 26/07/04 | GDC – Limerick |
| System Architect | N/A | N/A | N/A |
| Tech Lead | Sandro Cotliarenko | 27/07/04 | GDC – Nashville |
| Test Lead | Fabricio Silva | 26/07/04 | GDC – Limerick |
| Caso 2 | GFS – Quarterly Enhancement | | |
| Entrevistado | Nome | Data | Local |
| Delivery Manager | Felipe Soares | 05/08/04 | GDC – Nashville |
| Project Manager | Geraldo Gomes | 27/07/04 | GDC – Limerick |
| System Architect | Cristina Martini | 04/08/04 | GDC – Penang |
| Tech Lead | N/A | N/A | N/A |
| Test Lead | Daniele Araujo | 12/08/04 | GDC – Bray |

| H. Análise dos dados | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 13/08/2004 – 30/08/2004 |

**2. Escolha dos respondentes**

| Respondentes | Dimensões | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| Delivery Manager | X | | X | X | X | X |
| System Architect | X | X | X | X | X | X |
| Project Manager | X | X | X | X | X | X |
| Tech Leader | X | X | | X | X | X |
| Test Leader | X | X | | X | X | X |

**3. Outros recursos utilizados**

A. Recursos materiais
- → Um micro-gravador;
- → Uma sala de reuniões na organização;
- → Um microcomputador.

**4. Modelo do estudo e Dimensões da Pesquisa**

A Figura 31 representa graficamente os principais aspectos enfocados no desenvolvimento deste trabalho.

**5. Análise de dados**

Esta entrevista insere-se em uma pesquisa de base qualitativa, exploratória, baseada na técnica de análise de conteúdo, aplicado conforme proposto por Yin (2001). A coleta de dados envolve fontes primárias (resultado da aplicação do instrumento) e fontes secundárias (documentação e registros de arquivos). A triangulação dos dados coletados permitirá maior confiabilidade nos resultados obtidos.

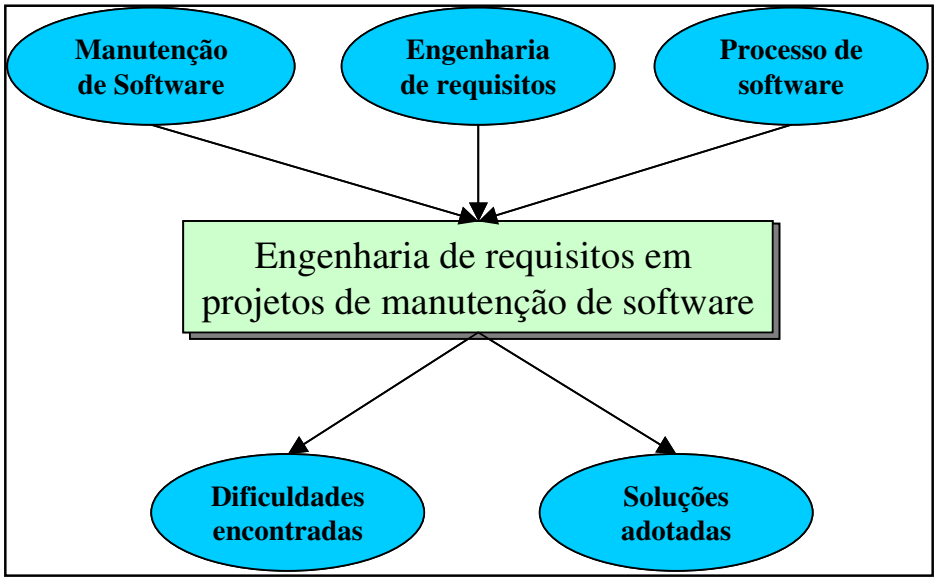


**Figura 31 - Aspectos enfocados no trabalho**

**Roteiro das Entrevistas**

**Dimensão 1 – Dados Demográficos**

Identificação pessoal

1. Nome:
2. Idade: anos

3. Escolaridade (Informe somente o maior grau)

| | | |
|---|---|---|
| ( ) – 1º Grau | ( ) – Superior Completo | ( ) – Mestrado Completo |
| ( ) – 2º Grau | ( ) – Especialização | ( ) – MBA Incompleto |
| ( ) – Superior Incompleto | ( ) – Mestrado Incompleto | ( ) – MBA Completo |

Responda as questões 4, 5 e 6 conforme o grau de escolaridade assinalado na questão 3.

4. Curso:
5. Universidade:
6. Ano de conclusão:

7. Tempo de experiência profissional na área de informática: __anos
8. Tempo de experiência profissional em projetos de manutenção de software: _anos
9. Empresa
   ( ) – Dell
   ( ) – Terceiros
   ( ) – PUCRS ( estagiário )
10. Tempo de empresa: __anos
11. Função (Conforme terminologia utilizada na empresa):

**Dimensão 2 – Processo de Manutenção de Software**

12. Existe um processo definido para a manutenção de software na empresa? Como é chamado [IEE98] [ISO98] [ISO99]?
13. Descreva o processo utilizado no seu projeto em termos dos seguintes componentes [IEE98] [ISO98] [ISO99]:
    a. Ciclo de vida?
    b. Papeis?
    c. Atividades?
    d. Artefatos?
14. Que artefatos são recebidos como entrada para o processo de manutenção de software [IEE98] [ISO99]?
15. Que artefatos são gerados como saída do processo de manutenção de software [IEE98] [ISO99]?
16. Quais dos seguintes tipos de manutenção o seu projeto está atendendo [ISO99]?

| **Tipo** | **Definição em [ISO99]** |
|---|---|
| ( ) – Corretiva | Modificação reativa de um produto de software realizada para corrigir falhas descobertas após sua entrega. |
| ( ) – Preventiva | Modificação em um produto de software, após sua entrega ao cliente, para detectar e corrigir falhas latentes antes que elas se tornem falhas efetivas. |
| ( ) – Adaptativa | Modificação realizada em um produto de software, após sua entrega, para manter o programa funcionando em um ambiente diferente ou que esteja mudando. |
| ( ) – Perfectiva | Manutenções que visam otimizar características não-funcionais, como performance ou facilidade de manutenção de um software ou acrescentar novas funcionalidades. |

( ) – Nenhuma das anteriores. Qual?

**Dimensão 3 – Gerência de Projetos**

17. Considerando os seguintes tipos de manutenção, quais são suportados pelo processo atual de manutenção de software [ISO99]?
    ( ) – Corretiva — Modificação reativa de um produto de software realizada para corrigir falhas descobertas após sua entrega.
    ( ) – Preventiva — Modificação em um produto de software, após sua entrega ao cliente, para detectar e corrigir falhas latentes antes que elas se tornem falhas efetivas.
    ( ) – Adaptativa — Modificação realizada em um produto de software, após sua entrega, para manter o programa funcionando em um ambiente diferente ou que esteja mudando.
    ( ) – Perfectiva — Manutenções que visam otimizar características não-funcionais, como performance ou facilidade de manutenção de um software ou acrescentar novas funcionalidades.
    ( ) – Nenhuma das anteriores. Qual?
18. Baseado em sua experiência na empresa, qual a proporção de projetos realizados para cada tipo de manutenção de software [ISO99]?
    a. Corretiva ___%
    b. Preventiva ___%
    c. Adaptativa ___%
    d. Perfectiva ___%
    e. Nenhuma das anteriores. Qual? ___%
19. Qual é a estrutura das equipes de manutenção de software em termos de papéis e responsabilidades [IEE98] [ISO99]?

**Dimensão 4 – Engenharia de Requisitos**

20. Na sua opinião, o que é um requisito [KOT98]?
21. Existe alguma técnica formal de engenharia de requisitos sendo utilizada pela empresa? Como é chamada [KOT98]?
22. Descreva o processo de engenharia de requisitos em termos dos seguintes componentes [KOT98] [ISO98]:
    a. Ciclo de vida?
    b. Papeis?
    c. Atividades?
    d. Artefatos?
23. Você é responsável por quais das seguintes atividades do processo de engenharia de requisitos [KOT98]?
    ( ) – Elicitação.
    ( ) – Análise e negociação.
    ( ) – Documentação.
    ( ) – Validação.
    ( ) – Gerência.
    ( ) – Nenhuma das anteriores. Qual?
24. Quais são os artefatos recebidos como entrada para a engenharia de requisitos [KOT98]?
25. São recebidos os requisitos anteriores do software, ou seja, tanto os requisitos originais (do projeto de desenvolvimento) quanto os requisitos estabelecidos em todos os projetos de manutenção anteriores [KOT98] [LIU99] [THA90] [ZAN03]?
26. Como são documentadas as alterações nos requisitos anteriores do software [KOT98] [LIU99] [THA90] [ZAN03]?
27. Quais requisitos são considerados na elaboração dos planos de teste [KOT98] [LIU99] [THA90] [ZAN03]?
    ( ) – Apenas os requisitos novos ou alterados.
    ( ) – Apenas os requisitos anteriores.
    ( ) – Tanto os requisitos antigos quanto os novos ou alterados.
28. Quais são os artefatos gerados como saída pela engenharia de requisitos [KOT98]?

**Dimensão 5 – Dificuldades**

29. Quais foram as dificuldades que você já encontrou na engenharia de requisitos em projetos de manutenção de software?
30. Em que atividades estas dificuldades foram encontradas?
31. Para cada dificuldade apontada, quais foram as soluções adotadas?
32. Como estas soluções foram identificadas?
33. Na sua analise, qual foi o impacto da solução adotada, tanto para o projeto quanto para a empresa?

**Dimensão 6 – Observações Gerais**

34. Você tem algum comentário adicional ou gostaria de acrescentar algo ao que foi dito nesta entrevista?

# APÊNDICE B – Protocolo de pesquisa do estudo de caso 2

## Avaliação de um modelo de arquitetura de informação para gerência de requisitos em desenvolvimento distribuído de software

**Objetivo**
Avaliar a aplicabilidade de um modelo de estrutura para gerência de requisitos em desenvolvimento distribuído de software (DDS) e os benefícios decorrentes de sua utilização em projetos reais conduzidos em um ambiente DDS.

**Questão de pesquisa**
Como desenvolver a gerência de requisitos em projetos conduzidos em ambientes de desenvolvimento distribuído de software?

**Unidade de estudo**
Projetos de manutenção de software da organização.

**Característica-chave do método de estudo de caso**
Este é um roteiro para o desenvolvimento e aplicação de um instrumento de pesquisa estruturada com questões em escala Likert que se caracteriza como uma pesquisa do tipo transeccional.

**Organização deste protocolo**
Este protocolo está organizado como segue:

6. **Procedimentos**

| A. Reuniões para levantamento das questões e estruturação do roteiro de entrevista | |
|---|---|
| Participantes: | Rodrigo Santos de Espindola<br>Prof. Dr. Jorge Luis Nicolas Audy. Doutor em Sistemas de Informação – UFRGS - 2001 (Especialista, Pesquisador Sênior) |
| Local: | PRPPG – Pró-reitoria de Pesquisa e Pós-Graduação |
| Data: | 24/11/2005 |

| B. Validação de face e conteúdo | |
|---|---|
| Participante: | Prof. Dr. Ricardo Mello Bastos. Doutor em Ciência da Computação – UFRGS 1998 (Especialista, Pesquisador Sênior).<br>Prof. Dr. Marcelo Blois. Doutor em Ciência da Computação – PUC-RJ – 2002 (Especialista, Pesquisador Sênior).<br>Rodrigo Santos de Espindola |
| Locais: | AGT – Agencia de gestão tecnológica.<br>PPGCC – Programa de Pós-graduação em Ciência da Computação. |
| Datas: | 01/12/2005 e 02/12/2005 |

| C. Adequação do roteiro de entrevista com base na validação de face e conteúdo | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 02/12/2005 |

| D. Pré-teste | |
|---|---|
| Participantes: | Rodrigo Santos de Espindola<br>Elson Zago – Tech Leader |
| Local: | Dell GDC – Global Development Center |
| Data: | 03/12/2005 |
| Participantes: | Rodrigo Santos de Espindola<br>Azriel Majdenbaum – System Architect |
| Local: | Dell GDC – Global Development Center |
| Data: | 03/12/2005 |

| E. Adequação do roteiro de entrevista com base no pré-teste | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | 03/12/2005 |

| F. Autorização da empresa participante | |
|---|---|
| Participantes: | Ana Trindade (Dell Computadores do Brasil) |
| Local: | Autorizado por e-mail |
| Data: | 04/12/2005 |

| G. Análise dos dados | |
|---|---|
| Participante: | Rodrigo Santos de Espindola |
| Local: | CDPe – Centro de desenvolvimento e pesquisa Dell/PUCRS |
| Data: | TBD |

**7. Escolha dos respondentes**

| Respondentes | Dimensões | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Gerente de projeto | X | X | X | X |
| Engenheiro de Requisitos | X | X | X | X |
| Lider de desenvolvimento | X | X | X | X |
| Lider de testes | X | X | X | X |

**8. Outros recursos utilizados**

A. Recursos materiais

→ Sistema de gerenciamento de e-mails para envio e recepção das entrevistas.

**9. Modelo do estudo e Dimensões da Pesquisa**

A Figura 32 representa graficamente os principais aspectos enfocados no desenvolvimento desta pesquisa.

**10. Análise de dados**

Esta entrevista insere-se em uma pesquisa de base qualitativa, exploratória, baseada na técnica de análise de conteúdo, aplicado conforme proposto por Yin **Erro! Fonte de referência não encontrada.**, e o módulo estatístico do Excel. A coleta de dados envolve fontes primárias (resultado da aplicação do instrumento) e fontes secundárias (documentação e registros de arquivos). A triangulação dos dados coletados permitirá maior confiabilidade nos resultados obtidos.

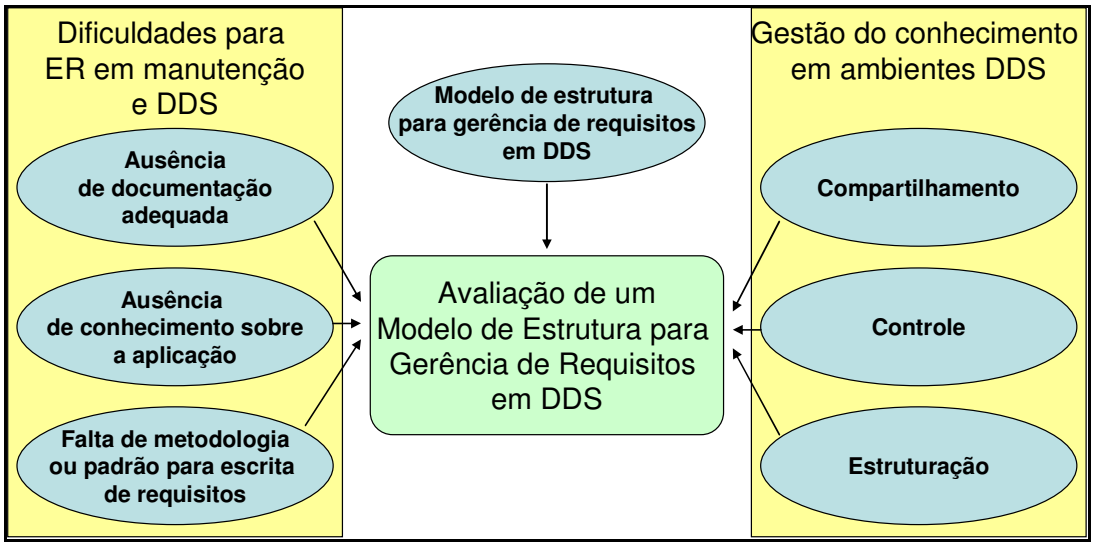


**Figura 32 - Aspectos enfocados na pesquisa**

**Roteiro das Entrevistas**

**Dimensão 1 – Dados Demográficos**

Identificação pessoal

1. Nome:
2. Idade: __anos

3. Escolaridade (Informe somente o maior grau)

( ) – 1º Grau ( ) – 2º Grau ( ) – Especialização
( ) – Superior Completo ( ) – Superior Incompleto
( ) – Mestrado Completo ( ) – Mestrado Incompleto

Responda as questões 4, 5 e 6 conforme o grau de escolaridade assinalado na questão 3.

4. Curso:
5. Universidade:
6. Ano de conclusão:

7. Tempo de experiência profissional em funções relacionadas a área de tecnologia da informação: __anos

Baseado em sua experiência na aplicação do modelo de estrutura para gerência de requisitos em desenvolvimento distribuído de software, preencha o formulário abaixo assinalando com um "X" a opção mais condizente com o seu grau de concordância com as afirmações propostas.

**Dimensão 2 – Dificuldades**

| **Aspectos** | **Questão** |
|---|---|
| ER em manutenção de software **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** | 8. O modelo permite reduzir as dificuldades relativas à ausência de documentação em projetos de manutenção.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 9. O modelo contribui para a preservação da documentação sobre requisitos ao longo de diversos projetos.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| ER em Desenvolvimento Distribuído de Software **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** | 10. O modelo facilita o acesso, em um ambiente DDS, às informações sobre os requisitos do software.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 11. O modelo facilita o gerenciamento da documentação sobre requisitos do software em ambientes DDS.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 12. O modelo contribui para a preservação do conhecimento sobre a aplicação.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |

**Dimensão 3 – Aplicabilidade**

| Aspectos | Questão |
|---|---|
| Gerência de Requisitos **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** | 13. O modelo é de fácil adoção.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 14. O modelo pode ser empregado independentemente da técnica de especificação de requisitos adotada.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 15. O modelo é independente das ferramentas utilizadas para sua implantação.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 16. O modelo torna o processo de gerência de requisitos significativamente mais caro.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 17. O modelo reduz significativamente a eficiência do processo de gerência de requisitos.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 18. O modelo contribui para o incremento da qualidade das especificações de requisitos.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 19. O modelo torna o processo de gerência de requisitos significativamente mais complexo.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |

**Dimensão 4 – Gestão do conhecimento**

| Aspectos | Questão |
|---|---|
| Compartilhamento **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** | 20. O modelo facilita a transferência de conhecimento sobre requisitos entre equipes geograficamente dispersas.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 21. O modelo facilita o compartilhamento do conhecimento sobre requisitos entre equipes geograficamente dispersas.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| Controle **Erro! Fonte de referência não encontrada.** | 22. O modelo facilita o controle, por parte dos diversos times geograficamente distribuídos, do conhecimento sobre os requisitos.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| Estruturação **Erro! Fonte de referência não encontrada.** | 23. O modelo reduz a fragmentação da documentação sobre os requisitos do software.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 24. O modelo reduz a dispersão das informações sobre os requisitos do software.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 25. O modelo contribui para a padronização da documentação de requisitos.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |

**Dimensão 5 – Percepção geral**

| Aspectos | Questão |
|---|---|
| Benefícios | 26. Em termos gerais, o uso do modelo foi benéfico para o projeto.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |
| | 27. Em termos gerais, foi mais fácil gerenciar os requisitos com o uso do modelo.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |

| | |
|---|---|
| Generalização | 28. O modelo pode ser aplicado tanto em projetos de desenvolvimento quanto em projetos de manutenção de software.<br>Discordo totalmente ☐ ☐ ☐ ☐ ☐ Concordo Totalmente |

# APÊNDICE C – Artigos publicados

Com o objetivo de submeter os resultados desta pesquisa à apreciação da comunidade científica, foram desenvolvidos durante os anos de 2004 e 2005 artigos para eventos relacionados às áreas de engenharia de requisitos e manutenção de software. Dos quatro artigos submetidos, três foram aceitos, publicados e apresentados. A seguir, são descritos brevemente estes artigos:

Espindola, R. S.; Majdenbaum A.; Audy, J. L. N. "Uma Análise Crítica dos Desafios para Engenharia de Requisitos em Manutenção de Software". In: VII Workshop on Requirements Engineering. Tandil, Argentina, 2004.
Objetivo: Apresentar uma análise crítica da influência das principais dificuldades encontradas na manutenção de sistemas legados sobre os processos da Engenharia de Requisitos.

Espindola, R. S.; Majdenbaum A.; Audy, J. L. N. "Requirements Engineering Challenges for Software Maintenance Projects in Distributed Software Development Environments". 21st International Conference on Software Maintenance. Budapest, Hungria, 2005.
Objetivo: Analisar os desafios encontrados na ER para projetos de manutenção de software em ambientes de DDS.

Espindola, R. S.; Lopes, L. T.; Prikladnicki, R.; Audy, J. L. N. "Uma Abordagem Baseada em Gestão do Conhecimento para Gerência de Requisitos em Desenvolvimento Distribuído de Software". In: VIII Workshop on Requirements Engineering. Porto, Portugal, 2005.
Objetivo: Analisar a engenharia de requisitos em projetos de desenvolvimento distribuído de software sob a ótica da gestão de conhecimento.

Esta pesquisa foi parcialmente financiada pelo Convênio Dell/PUCRS, através da Lei de Informática Brasileira (Lei nº. 8.248/91).

Este trabalho foi digitado conforme o Guia para Apresentação de Trabalhos Acadêmicos, Teses e Dissertações da Biblioteca Central Irmão José Otão da PUCRS, segundo a NBR 14724 proposta pela ABNT, atualizado em 30 de janeiro de 2006.