

ANA MARIA PEREIRA

**ABORDAGEM DE ESPECIFICAÇÃO DE REQUISITOS BASEADA EM
PROJETO AXIOMÁTICO**

Trabalho de Dissertação apresentado ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Mestre em Ciências - Área de Concentração: Engenharia de Computação
Orientador: Prof. Dr. Paulo César Stadzisz

CURITIBA

2011

Dados Internacionais de Catalogação na Publicação

P436 Pereira, Ana Maria
Abordagem de especificação de requisitos baseada em projeto axiomático / Ana
Maria Pereira
.— 2011.
168 p. : il. ; 30 cm

Orientador: Paulo César Stadzisz.

Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de
Pós-graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2011.

Bibliografia: p. 149-153.

1. Engenharia de sistemas. 2. Engenharia de software. 3. Software – Desenvolvimento.
4. Modelagem (Computação). 5. UML (Linguagem de modelagem padrão). 6. Software –
Confiabilidade. 7. Teoria axiomática dos conjuntos. 8. Método orientado a objetos
(Computação). 9. Engenharia elétrica – Dissertações. I. Stadzisz, Paulo César, orient. II.
Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em
Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

Biblioteca Central da UTFPR, Campus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

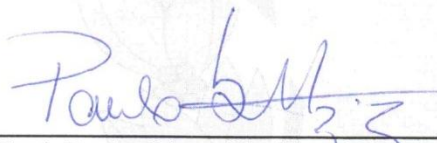
Título da Dissertação Nº 572:

**“Abordagem de Especificação de Requisitos
Baseada em Projeto Axiomático”**

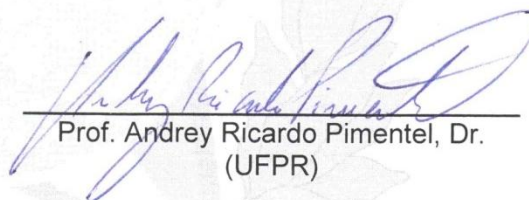
por

Ana Maria Pereira

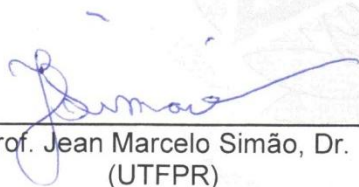
Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Engenharia de Automação e Sistemas, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR – Campus Curitiba, às 14h30 do dia 25 de agosto de 2011. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:



Prof. Paulo César Stadzisz, Dr.
(Presidente – UTFPR)



Prof. Andrey Ricardo Pimentel, Dr.
(UFPR)



Prof. Jean Marcelo Simão, Dr.
(UTFPR)

Visto da coordenação:



Prof. Fábio Kurt Schneider, Dr.
(Coordenador do CPGEI)

Resumo

PEREIRA, A. M. Abordagem de especificação de requisitos baseada em Projeto Axiomático. 2011. 168 f. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Programa de Pós Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2011.

Esta dissertação apresenta uma abordagem que aplica a Teoria de Projeto Axiomático à especificação de requisitos de sistemas de *software* visando melhorar a qualidade da solução de projeto desde a análise do problema até identificação dos requisitos efetivamente. O objetivo da abordagem desenvolvida é propor e integrar métodos que permitam o uso de Projeto Axiomático em um processo de Engenharia de Requisitos. A abordagem de especificação de requisitos proposta estabelece a aplicação do Axioma da Independência no estudo de problemas e necessidades do cliente. Desta forma, incluem-se no processo de engenharia de requisitos novos domínios de estudo, o domínio do problema e o domínio do cliente. É estabelecido um modelo de hierarquia para a decomposição de Problemas, Necessidades e Requisitos. Um processo de *zigzagamento* é sugerido para que se possa aplicar a abordagem proposta em conjunto com um processo de desenvolvimento iterativo e incremental como o Processo Unificado. Apresenta-se um caso de estudo de um sistema de teste de equipamentos em uma linha de produção. O caso de estudo tem por objetivo demonstrar a aplicação prática da abordagem de especificação de requisitos proposta nesta dissertação. Além disso, apresentam-se alguns experimentos realizados durante o trabalho de pesquisa e seus resultados. Para exemplificar a realização dos experimentos é demonstrado o processo de especificação de requisitos para um sistema de relatórios de visitas a clientes. Este exemplo ajuda a ilustrar como a abordagem apresentada pode ser utilizada para aumentar a consistência e a qualidade dos requisitos de projetos de *software*.

Palavras-chave: Requisitos, Necessidades do Cliente, Problema, Projeto Axiomático, Matriz de Rastreabilidade, Especificação de Requisitos.

Abstract

PEREIRA, A. M. Requirements specification approach based on Axiomatic Design. 2011. 168 f. Dissertation (Master in Electrical Engineering and Industrial Informatics) – Post Graduate Program in Electrical Engineering and Industrial Informatics, Federal Technological University of Parana, Curitiba, 2011.

This dissertation presents an approach that applies the Axiomatic Design Theory to the specification of software systems requirements. This approach intends to improve the quality of design solution since its inception, which involves from the problem analysis to the requirements identification. The purpose of the proposed approach is to offer methods that allow the use of axiomatic design in a process of requirements engineering. The proposed requirements specification approach establishes the application of the Axiom of Independence in the study of problems and customer needs. In this way, new domains, the problem domain and the customer domain, are included in the requirements engineering process. It is established a hierarchical model for the decomposition of Problems, Needs and Requirements. A *zig-zag* process is suggested in order to use the propose approach in conjunction with a development process as the Unified Process. It is presented a case study of a system for equipment testing in a production line. The case study aims to demonstrate the practical application of the requirements specification approach proposed in this dissertation. In addition, the results of the experiments performed during the research are presented. The requirements specification process for a reporting system is shown to illustrate the experiments. This example helps illustrate how the proposed approach can be used to increase the consistency and quality of software requirements.

Keywords: Requirements, Customer Needs, Problems, axiomatic design, Traceability Matrix, Requirements Specification

Lista de Figuras

Figura 1 - Método de Trabalho	24
Figura 2 - Disciplinas de Engenharia de Sistemas	29
Figura 3 - Relação UML e SysML	31
Figura 4 - Diagramas da SysML.....	32
Figura 5 - Diagrama de Requisitos.....	33
Figura 6 - Engenharia de <i>Software</i> em Camadas	34
Figura 7 - Estrutura do Processo Unificado.....	39
Figura 8 - Fluxo de Requisitos do Processo Unificado.....	45
Figura 9 - Modelo das Pirâmides Gêmeas	48
Figura 10 - Níveis de Maturidade de Requisitos.....	50
Figura 11 - Desempenho de Projetos por Nível de Maturidade de Requisitos	51
Figura 12 - Pirâmide das Necessidades de Maslow.....	53
Figura 13 - Hierarquia das Necessidades Organizacionais.....	56
Figura 14 - Custo da Má Especificação de Requisitos	64
Figura 15 - Custo de Correção de Defeitos em Diferentes Fases do Projeto.....	64
Figura 16 - Domínios de Projeto.....	69
Figura 17 - Processo em Zig-zag	69
Figura 18 - Matriz de Projeto Desacoplada	70
Figura 19 - Matriz de Projeto Semiacoplada	71
Figura 20 - Matriz de Projeto Acoplada	72
Figura 21 - Processo de Desenvolvimento Axiomático de <i>Software</i>	74
Figura 22 - Realização de Casos de Uso	77
Figura 23 - Decomposição de Casos de Uso e Colaborações	77
Figura 24 - Domínios de Projeto Axiomático e as Fases do Processo Unificado	79
Figura 25 - Exemplo de Matriz de Projeto Desacoplada	81
Figura 26 - Visão de Domínios de Problema e Solução.....	89
Figura 27 - Nova Visão de Domínios de Problema e Solução.....	90
Figura 28 - Relacionamento Entre Domínios.....	91
Figura 29 - Domínios de projeto	92
Figura 30 - Projeto Axiomático e fases do Projeto Unificado	93
Figura 31 - Fluxo de Requisitos do PU e Abordagem Proposta	94
Figura 32 - Níveis de Detalhamento	96
Figura 33 - Rastreabilidade Bidirecional.....	98
Figura 34 - Matriz de Necessidades X Requisitos.....	99
Figura 35 - Análise do Problema	100
Figura 36 - Elementos UML Representando Elementos dos Domínios	103
Figura 37 - Relacionamento Entre Elementos.....	105
Figura 38 - Diagrama SDT das Etapas da abordagem	107
Figura 39 – Diagrama de Atividades Modelo Proposto	110
Figura 40 - Atividades com Foco no Problema.....	114
Figura 41 - Atividades com Foco nas Necessidades.....	115
Figura 42 - Atividades com Foco nos Requisitos	116
Figura 43- Tipos de Relacionamentos em uma Matriz de Projeto.....	117
Figura 44 - Relacionamento Acoplado	118
Figura 45 - Tela Principal	120

Figura 46 - Diagrama Problema X Necessidades, Etapa 1	125
Figura 47 - Matriz de projeto, Etapa 1	125
Figura 48 - Detalhamento dos problemas, Etapa 1	127
Figura 49 - Detalhamento das Necessidades, Etapa 2	128
Figura 50 – Necessidades X Requisitos, Etapa 2	128
Figura 51 - Matriz de projeto, Etapa 2	129
Figura 52 - Detalhamento de Necessidades, Etapa 3	130
Figura 53 - Características X Sub-Requisitos Essenciais, Etapa 3	131
Figura 54 - Matriz de Projeto, Etapa 3.....	131
Figura 55 - Requisitos X Casos de Uso, Etapa 3	132
Figura 56 - Detalhamento de Requisitos, Etapa 4.....	133
Figura 57 - Problemas X Necessidades da Primeira Etapa	138
Figura 58 - Necessidades X Requisitos da Primeira Etapa	139
Figura 59 - Problemas X Necessidades da Segunda Etapa	141
Figura 60 - Necessidades X Requisitos da Segunda Etapa	142
Figura 61 - Cálculo da Reangularidade e Semangularidade	157
Figura 62 - Tela Inicial.....	158
Figura 63 - Cadastrar Elemento	158
Figura 64 - Mensagem de Alerta	159
Figura 65 - Adicionar Elemento com Relacionamentos.....	160
Figura 66 - Matriz de Projeto	161

Lista de Tabelas

Tabela 1 - Distribuição de Resultados de Projetos.....	18
Tabela 2 - Distribuição dos Fatores de Falhas em Projetos.....	63
Tabela 3 - Níveis de Abstração	78
Tabela 4 - Métricas de Complexidade	81
Tabela 5 - Níveis de Abstração	101
Tabela 6 - Lista de Problemas da Primeira Etapa.....	137
Tabela 7 - Lista de Necessidades da Primeira Etapa.....	137
Tabela 8 - Lista de Requisitos da Primeira Etapa	138
Tabela 9 - Motivação das Necessidades.....	139
Tabela 10 - Problemas Reformulados.....	140
Tabela 11 - Necessidades Reformuladas	140
Tabela 12 - Novo Conjuntos de Requisitos	141

Lista de Abreviaturas, Siglas e Acrônimos

AI	Axioma da Independência
Alnf	Axioma da Informação
CMMI	<i>Capability Maturity Model Integration</i>
EA	<i>Enterprise Architect</i>
ES	Engenharia de <i>Software</i>
ESis	Engenharia de Sistemas
ER	Engenharia de Requisitos
EROO	Engenharia de Requisitos Orientada a Objetivos
IRMM	<i>IAG Requirement Maturity Model</i>
MIT	<i>Massachusetts Institute of Technology</i>
MPSBR	Melhoria de Processo de <i>Software</i> Brasileiro
NC	Necessidade do Cliente
NH	Necessidade Humana
NO	Necessidade Organizacional
PA	Projeto Axiomático
PB	Problemas
PP	Parâmetro de Projeto
POO	Programação Orientada a Objetos
RMM	Requirement Maturity Model
RQ	Requisito
RF	Requisito Funcional
SysML	<i>Systems Modeling Language</i>
TPA	Teoria de Projeto Axiomático
UML	<i>Unified Modeling Language</i>

Sumário

1	Introdução	17
1.1	Contexto e Motivação	17
1.1.1	Motivação e Justificativa	20
1.2	Objetivos	21
1.3	Método	22
1.4	Organização do Trabalho.....	24
2	Fundamentação Teórica	27
2.1	Engenharia de Sistemas	27
2.1.1	Visão Geral de Engenharia de Sistemas	27
2.1.2	Modelagem de Sistemas com SysML.....	30
2.2	Engenharia de <i>Software</i>	33
2.2.1	Visão Geral da Engenharia de <i>Software</i>	34
2.2.2	Projeto de <i>Software</i>	35
2.2.3	Processo de <i>Software</i>	36
2.2.4	Modelagem de <i>Software</i> com a UML.....	40
2.3	Engenharia de Requisitos	43
2.3.1	Visão Geral de Engenharia de Requisitos	43
2.3.2	Processo de Engenharia de Requisitos	44
2.3.3	Abordagem de Especificação de Requisitos Orientada a Objetivos	46
2.3.4	Quadros de Problema.....	47
2.3.5	Modelo de Maturidade de Requisitos.....	49
2.4	Conceitos Relacionados à Especificação de Requisitos.....	51
2.4.1	Necessidades Humanas	52
2.4.2	Necessidades Organizacionais.....	54
2.4.3	Problema	56
2.4.4	Domínio do Problema	57
2.4.5	Necessidades do Usuário	58
2.4.6	Características do Sistema	59
2.4.7	Requisitos de Sistema	60
2.5	Problemas Relacionados a Projetos de <i>Software</i>	61
2.6	Conclusões	65
3	Projeto Axiomático	67
3.1	Introdução à Teoria de Projeto Axiomático	67
3.2	Domínios de Projeto	68
3.3	Axioma da Independência.....	70
3.4	Axioma da Informação	73
3.5	Projeto Axiomático e Desenvolvimento de <i>Software</i>	73
3.6	Desenvolvimento Orientado a Objetos Baseado em Projeto Axiomático.....	75
3.6.1	Aspectos Gerais.....	75
3.6.2	Domínios de Projeto e Fases do Processo Unificado	78
3.6.3	Aplicação do Axioma da Independência	80
3.6.4	Aplicação do Axioma da Informação.....	81
3.7	Conclusões	83
4	Especificação de Requisitos Baseada no Projeto Axiomático	85
4.1	Introdução à Abordagem Proposta	85
4.2	Domínios do Desenvolvimento de Sistemas de <i>Software</i>	88

4.3 Projeto Axiomático e Processo Unificado	93
4.3.1 Mapeamento e Níveis de Detalhamento	95
4.4 Projeto Axiomático e Rastreabilidade de Requisitos	97
4.5 Conceitos Empregados na Abordagem Proposta	100
4.6 Etapas do Modelo	106
4.7 Atividades do Modelo Proposto	109
4.8 Atividades de cada Etapa	114
4.9 Princípios de Análise	117
4.10 Protótipo da Ferramenta	120
4.11 Conclusão	121
5 Caso de estudo	123
5.1 Objetivos do Caso de estudo	123
5.2 Descrição do Sistema	123
5.3 Descrição do Caso de estudo	124
5.3.1 Etapa 1 – Foco no problema	124
5.3.2 Etapa 2 – Foco nas Necessidades	127
5.3.3 Etapa 3 – Foco nos Requisitos Essenciais	129
5.3.4 Etapa 4 – Foco nos Requisitos Técnicos	133
5.4 Conclusão	134
6 Experimentos	135
6.1 Objetivos dos Experimentos	135
6.2 Metodologia dos Experimentos	135
6.3 Descrição do Problema	136
6.4 Primeira Etapa	137
6.5 Segunda etapa	139
6.6 Conclusões	142
7 Conclusões e Trabalhos Futuros	145
7.1 Conclusões	145
7.2 Trabalhos futuros	147
Referências	149
Apêndice A - Protótipo da Ferramenta	155
Anexo A – Teoremas e Corolários Relacionados com os Axiomas	163
A.1 Corolários	163
A.2 Teoremas Gerais de Projeto	164
A.3 Teoremas Relacionados com Projeto de <i>software</i>	168

1 Introdução

Este capítulo apresenta uma visão geral da dissertação por meio da descrição do tema do trabalho, do contexto no qual o tema está inserido e dos principais problemas que motivaram a realização do trabalho. Neste capítulo também são apresentados os objetivos gerais e específicos a serem alcançados, bem como a metodologia empregada no desenvolvimento da pesquisa de mestrado. Por fim, apresenta-se a organização dos capítulos deste documento.

1.1 Contexto e Motivação

Desde as primeiras décadas do surgimento da informática identificaram-se problemas relativos à elaboração e desenvolvimento de projetos de *software*. Logo no início dos anos 70 foi criado o termo “Crise de *Software*” para expressar as dificuldades enfrentadas pelos desenvolvedores na elaboração de seus projetos (DIJKSTRA, 1972). Naquela época, a engenharia de *software* era quase que inexistente e os problemas mais comuns costumavam ser: acréscimos no orçamento ou no prazo previsto; baixa qualidade dos projetos; produtos que muitas vezes não atendiam os requisitos; e códigos difíceis de manter.

Neste contexto, novos paradigmas, linguagens e abordagens foram desenvolvidos e, à medida que o *software* foi se tornando mais importante, novas metodologias e técnicas para seu desenvolvimento foram propostas. Estas técnicas e metodologias têm sido melhoradas continuamente com o passar do tempo. Cada nova metodologia representa um esforço para melhorar o processo de desenvolvimento de *software*.

Os programas de computador, atualmente, são aplicados em muitas áreas de atividade humana, como sistemas bancários, educação, comunicação, transportes e entretenimento, apenas para citar alguns exemplos. A demanda pelo desenvolvimento de *software* é crescente e o mercado muitas vezes não consegue suprir estas necessidades. Além disso, o resultado dos projetos frequentemente não é satisfatório, mesmo quando há disponibilidade de recursos e mão de obra.

Apesar das técnicas existentes, o grande desafio continua sendo construir sistemas de *software* com mais qualidade, dentro dos prazos e custos estabelecidos e, o mais importante, com as características esperadas destes sistemas. Estatísticas apresentadas pelo Chaos Report 2009 (DOMINGUEZ, 2009) demonstram que os projetos de *software* têm percentuais de sucesso que giram em torno de apenas 30% como apresenta a Tabela 1. Além disso, estudos relacionados à Engenharia de *Software* apontam fortes evidências de que os mais sérios problemas de projetos estão diretamente relacionados à definição e entendimento de requisitos (LEFFINGWELL; WIDRIG, 2003).

Tabela 1 - Distribuição de Resultados de Projetos

Distribuição de resultados de projetos de <i>software</i>	
Percentual	Resultados
32%	Sucesso (no prazo, dentro do orçamento e com escopo completo).
44%	Mudaram (atrasaram, estourou o orçamento, e/ou reduziram escopo).
24%	Falharam (cancelados ou nunca usados).

Fonte: Adaptado de (DOMINGUEZ, 2009)

Neste contexto, o principal desafio de um projeto de *software* está em especificar e construir um produto que garanta a satisfação das necessidades do cliente com alta qualidade e dentro dos prazos e custos estimados. Segundo Pressman (2006), os requisitos são fundamentais para a medição da qualidade de *software*. Assim, tendo em vista o desafio de atender as necessidades do cliente, procura-se garantir que todas elas sejam atendidas por meio da especificação e rastreamento de requisitos. Existem inúmeras abordagens de especificação de requisitos na literatura, dentre elas estão a Engenharia de Requisitos Orientada a Objetivos (van LAMSWEERDE, 2001), Quadros de Problemas (HALL et al., 2002) e a abordagem clássica de Engenharia de Requisitos utilizada, por exemplo, no Processo Unificado (KRUCHTEN, 2003).

A abordagem de Engenharia de Requisitos Orientada a Objetivos, também conhecida como KAOS (DARDENNE, van LAMSWEERDE, & FICKAS, 1993) propõem a identificação dos objetivos do sistema como forma de definir as características do sistema que será construído. Sendo identificados inicialmente alguns objetivos do sistema, o *framework* KAOS facilita a identificação de novos objetivos, requisitos e outras características do sistema (HEAVEN & FINKELSTEIN, 2004).

Por sua vez, o conceito de Quadros de Problema (*Problem Frames*) proposto por Jackson (1999) sugere o detalhamento dos problemas do cliente em subproblemas. O autor acredita que a melhor forma de realizar a análise de requisitos é por meio da abordagem de decomposição dos problemas. Sua intenção é analisar problemas de maneira mais realística decompondo-os em subproblemas que correspondem a Quadros de Problemas pré-estabelecidos (HALL et al., 2002).

O Processo Unificado é um processo de desenvolvimento de *software* amplamente conhecido, e é dito centrado na arquitetura e dirigido por casos de uso. Um de seus fluxos principais é o Fluxo de Requisitos, que não é em si uma técnica, mas um conjunto de atividades que tem por objetivo proporcionar ao desenvolvedor um entendimento dos requisitos do sistema (KRUCHTEN, 2003). Dentre as atividades do fluxo de requisitos está “Entender as Necessidades do Interessado”. Para que seja possível entender estas necessidades são sugeridas diversas técnicas, como entrevistas, *workshops* de requisitos, *brainstorming*, e *storyboards* (LEFFINGWELL & WIDRIG, 2003).

Em um trabalho anterior, realizado pelo grupo de pesquisa do CPGEI/UTFPR em que este trabalho foi desenvolvido, havia sido proposta uma abordagem de desenvolvimento de *software* que integra o Processo Unificado com a Teoria de Projeto Axiomático (PIMENTEL, 2007). A Teoria de Projeto Axiomático (SUH, 1990) surgiu como um processo para projeto de produtos, peças e componentes, usado em diversas áreas da engenharia. Este processo é baseado em conceitos de independência entre os requisitos funcionais do projeto e na minimização do conteúdo de informação do projeto. Esta teoria foi concebida inicialmente em 1978, para projetos de engenharia mecânica, principalmente na indústria automobilística (DEO & SUH, 2004), e muitos dos seus conceitos são aplicáveis diretamente ao projeto e produção. O Projeto Axiomático passou a ser aplicado em outras áreas como indústria eletrônica (DO & PARK, 1996) e manufatura (KULAK, DURMUSOGLU, & TUFEKI, 2005).

Os conceitos de Projeto Axiomático podem também ser aplicados no desenvolvimento de *software* e podem melhorar a qualidade do processo de desenvolvimento, garantindo a qualidade do produto desenvolvido. As metodologias e técnicas atuais de projeto de *software* possuem conceitos muito específicos. Uma metodologia de desenvolvimento de *software* está inserida em um processo. O processo de *software* mais usado atualmente é o Processo Unificado (BOOCH,

RUMBAUGH, & JACOBSON, 2006). Por esta razão, Pimentel (2007) propôs uma abordagem de Projeto de *Software* Orientado a Objetos Baseado na Teoria de Projeto Axiomático que se integra ao Processo Unificado.

Assim, continuando os estudos relacionados ao desenvolvimento de projetos de *software* orientado a objetos baseados em projeto axiomático iniciados por Pimentel (2007) optou-se por tratar nesta dissertação a questão da especificação de requisitos baseada também em projeto axiomático. Pretende-se, portanto, que a abordagem proposta possa ser utilizada de maneira integrada àquela sugerida primeiramente por Pimentel em seu trabalho.

1.1.1 Motivação e Justificativa

A motivação deste trabalho é contribuir para a melhoria da qualidade e produtividade no desenvolvimento de *software* por meio do aprimoramento do processo de Engenharia de Requisitos, aplicando a Teoria de Projeto Axiomático e fornecendo meios para a identificação das reais necessidades do cliente e dos requisitos adequados para atender estas necessidades. Assim, aumenta-se a chance de sucesso dos projetos em termos de atendimento das exigências e expectativas do mercado.

Nesta dissertação é proposta uma abordagem para a aplicação da Teoria de Projeto Axiomático na especificação de requisitos de sistemas de *software*. A Teoria de Projeto Axiomático é aplicada com sucesso a várias áreas do conhecimento humano. Esta teoria é objeto, inclusive, de diversos trabalhos na área de desenvolvimento de *software* como no trabalho de Pimentel (2007) que será apresentado mais adiante neste trabalho.

A maioria dos trabalhos que utilizam a Teoria de Projeto Axiomático está focada no estudo da aplicação dos axiomas na modelagem do projeto. A questão que este trabalho se propõe a discutir é se os axiomas de projeto aplicam-se às fases anteriores à modelagem. Isso significa, compreender se podem ser aplicados na especificação dos requisitos. A especificação de requisitos, neste caso, envolve o estudo dos problemas e necessidades do cliente como forma de identificar os requisitos mais adequados para o sistema.

Um dos principais objetivos deste trabalho foi estudar mais atentamente os problemas que os projetos de *software* pretendem resolver. Este trabalho foi realizado por acreditar-se que a chave para a identificação das necessidades clientes é a realização de um estudo detalhado de seus problemas. Além disso, propõe-se o uso de matrizes de rastreabilidade em conjunto com a aplicação do Axioma da Independência nestas matrizes como uma ferramenta para a elaboração de um projeto de *software* que resolva efetivamente os problemas do cliente.

É reconhecida a importância de se mapear o relacionamento entre requisitos e artefatos de *software* de maneira que haja rastreabilidade entre eles e que se garanta o atendimento de todos os requisitos. Nem tão conhecida é a técnica de se utilizar matrizes de rastreabilidade para mapear relacionamentos entre necessidades do cliente e requisitos. Além disso, apresentada-se neste trabalho, a proposta de se mapear relacionamentos entre problemas e necessidades do cliente utilizando matrizes de rastreabilidade. Para que seja possível este mapeamento, é necessário decompor o(s) problema(s) do cliente e em subproblemas. O que se pretende por meio deste estudo é demonstrar a aplicabilidade do axioma da independência proposto por Suh (1990) em seu modelo de Projeto Axiomático às matrizes de rastreabilidade, bastante conhecidas na Engenharia de *Software*.

Desta forma, por meio de uma abordagem de identificação e detalhamento dos problemas e necessidades do cliente, propõe-se oferecer uma ferramenta aos Engenheiros de Requisitos para enfrentar as dificuldades em compreender quais requisitos um sistema deve atender. Além disso, propõe-se a aplicação do Axioma da Independência no relacionamento entre os diversos domínios envolvidos em um projeto de *software* para ajudar a resolver problemas comuns de Engenharia de Requisitos, como inconsistências e requisitos incompletos.

1.2 Objetivos

Esta dissertação de mestrado aborda a integração entre a Teoria de Projeto Axiomático e métodos e técnicas de elaboração de requisitos de sistemas de *software*. O objetivo geral desta dissertação de mestrado é:

Desenvolver uma abordagem baseada na Teoria do Projeto Axiomático para a especificação e análise de requisitos de sistemas de *software*.

Os objetivos específicos deste trabalho são:

1. Definir correspondências entre os conceitos de SysML e UML (como requisitos, casos de uso, classes, colaborações) e os conceitos da teoria de Projeto Axiomático (como atributos do cliente, requisitos funcionais, parâmetros de projeto, restrições e variáveis de processo).
2. Estabelecer um padrão para um processo de “zigzagueamento” mais próximo de um ciclo de vida iterativo e incremental como o do Processo Unificado.
3. Sugerir melhorias nos processos de levantamento de requisitos por meio do estudo do processo de mapeamento de Problemas e Necessidades do cliente em Requisitos Funcionais e Casos de uso, sob a ótica do Projeto Axiomático.
4. Propor uma abordagem de especificação de requisitos que seja facilmente integrável à abordagem de Projeto de *Software* Orientado a Objetos baseado em Projeto Axiomático proposta por Pimentel (2007).
5. Realizar experimentos para verificar a aplicabilidade da abordagem proposta.
6. Desenvolver um caso de estudo para avaliação dos impactos da aplicação da abordagem em um projeto de *software*.
7. Desenvolver uma ferramenta protótipo que auxilie na verificação da Independência Funcional entre os elementos de cada domínio de projeto.

1.3 Método

Esta seção apresenta o método de trabalho aplicado nesta pesquisa de mestrado, que culminaram na elaboração deste documento. O método seguido é ilustrado pela Figura 1. Na primeira fase foi realizada a revisão inicial da literatura com objetivo de formar um conhecimento teórico sobre as técnicas de Engenharia de *Software* mais conhecidas na literatura e formar as bases deste estudo. Esta

etapa consistiu de estudos gerais sobre engenharia de *software*, processo unificado, metodologias ágeis, Projeto Axiomático, SysML e UML.

Na segunda fase foi realizada a revisão bibliográfica de assuntos específicos, que estavam diretamente relacionados ao tema proposto para a pesquisa de mestrado. O objetivo destes estudos foi identificar trabalhos que abordam o desenvolvimento de *software* seguindo a metodologia proposta pela teoria de projeto axiomático. Com base no trabalho desenvolvido por Pimentel (2007), foi elaborado o protótipo de uma ferramenta para o desenvolvimento de projetos de *software* orientados a objetos com projeto axiomático. Em concomitância com o desenvolvimento do protótipo, foi realizado um caso de estudo de um projeto de *software* seguindo a abordagem de Pimentel (2007). O objetivo destes trabalhos foi compreender melhor a abordagem e preparar para a quarta fase.

Na terceira fase, redação do Trabalho Individual, um documento de Trabalho Individual foi produzido, com base nos estudos realizados nas fases anteriores. O trabalho foi apresentado diante de uma banca de professores, por ser um requisito de qualificação do programa de mestrado e para que pudesse ser dado seguimento ao trabalho de pesquisa. Outro produto de trabalho desta fase do projeto foi o desenvolvimento de uma nova versão do protótipo da ferramenta para apoiar a aplicação da abordagem proposta. O protótipo desta ferramenta é apresentado no Apêndice A desta dissertação.

A quarta fase visou identificar possíveis melhorias na abordagem de Desenvolvimento de *Software* Orientado a Objetos Baseada em Projeto axiomático (PIMENTEL, 2007) e definir a abordagem que seria proposta no projeto de pesquisa. Após a definição, os elementos da abordagem foram descritos e especificados.

Na quinta fase, a Avaliação, realizaram-se experimentos com alguns voluntários que definiram problemas que gostariam de solucionar por meio de um sistema de *software*. Nestes experimentos foi aplicada a abordagem proposta. A partir destes experimentos foi possível elaborar estudos de caso utilizando a abordagem proposta. O objetivo destes estudos foi avaliar se a aplicação da abordagem é viável e quais benefícios traz para a especificação de requisitos e para o projeto como um todo. Além disso, o conhecimento adquirido fornece subsídios para o refinamento da abordagem e gerar hipóteses da sua aplicação a serem investigados em trabalhos de pesquisa futuros.

Finalmente, a sexta fase consistiu em produzir um artigo intitulado “Proposta de uma Abordagem para Especificação de Requisitos Baseada em Projeto Axiomático”, que foi aceito no III *Congreso Internacional de Computación y Telecomunicaciones* (COMTEL 2011) realizando em Lima, Peru de 19-21 de outubro de 2011, além da produção deste documento de dissertação e defesa da dissertação.

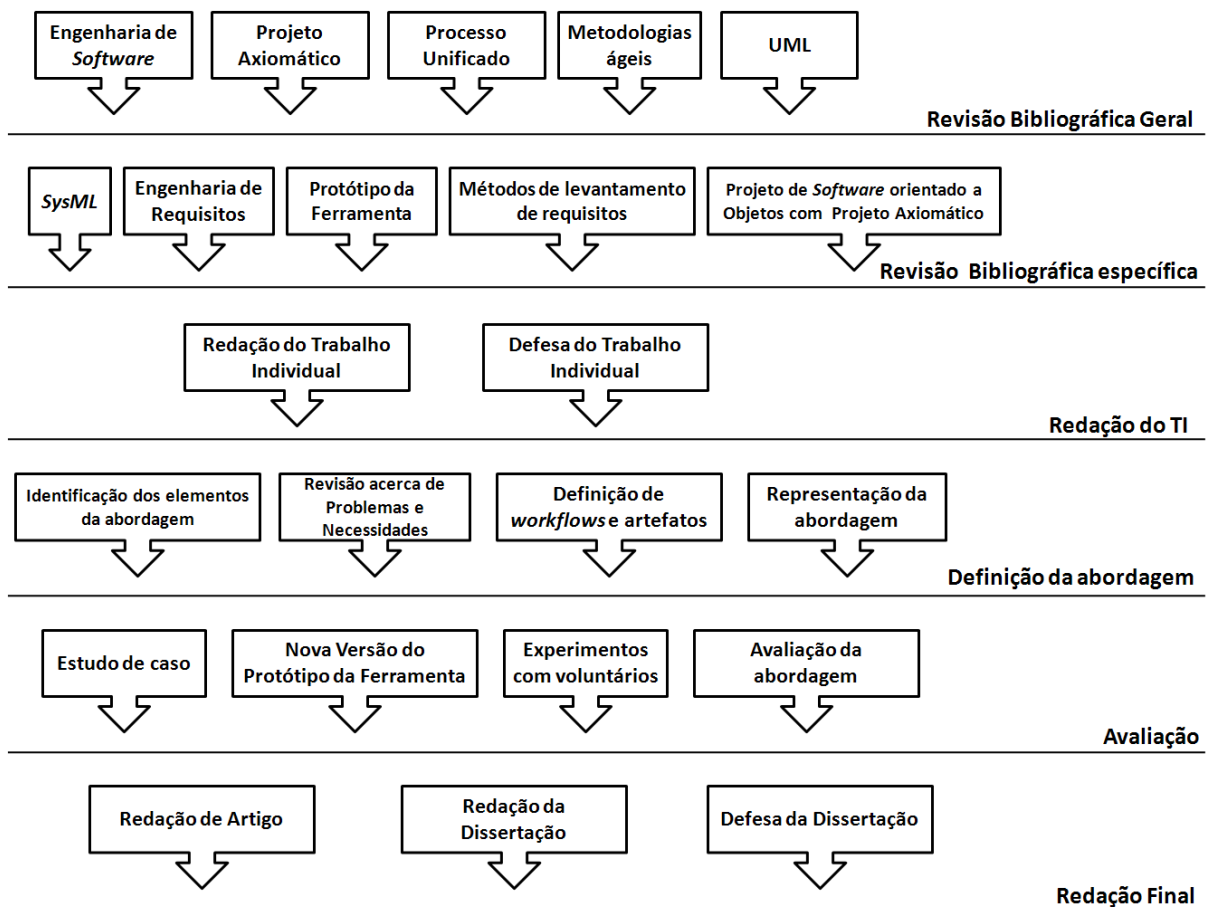


Figura 1 - Método de Trabalho
Fonte: Autoria própria

1.4 Organização do Trabalho

Neste capítulo foi apresentado o contexto em que este trabalho foi elaborado, apresentando alguns dos problemas recorrentes nos projetos de desenvolvimento de *software* no que se refere à especificação de requisitos de *software* adequados e de qualidade. Além disso, foi apresentada a motivação da

pesquisa realizada, os objetivos do trabalho e a metodologia que orientaram o desenvolvimento do mesmo.

Esta dissertação apresenta o trabalho de mestrado realizado no programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) da Universidade Tecnológica Federal do Paraná (UTFPR). Esta dissertação está organizada em sete capítulos, além da introdução.

O Capítulo 2 apresenta a fundamentação teórica deste trabalho abordando uma discussão sobre os principais problemas e dificuldades em projetos de sistemas de *software*, um estudo sobre Engenharia de Sistemas, um resumo dos principais conceitos de Engenharia de *Software*, um estudo das principais abordagens de Engenharia de Requisitos e um estudo sobre os conceitos de Problemas, Necessidades, Propriedades e Requisitos.

No Capítulo 3 é apresentado um resumo dos principais conceitos da Teoria de Projeto Axiomático. São apresentados os principais conceitos da Abordagem de projeto de *software* orientado a objetos baseado em Projeto Axiomático.

A abordagem para especificação de requisitos baseada em Projeto Axiomático proposta neste trabalho é apresentada no Capítulo 4. É apresentada uma visão geral da abordagem, seus conceitos e aplicações na especificação de requisitos.

No capítulo 5 é apresentado um caso de estudo com o objetivo de demonstrar a aplicação da abordagem proposta de especificação de requisitos.

No capítulo 6 são apresentados os experimentos realizados. Destes experimentos executados foi selecionado um exemplo para demonstrar a avaliação da abordagem proposta. Neste capítulo também é apresentada uma breve discussão sobre os resultados alcançados com os experimentos.

No capítulo 7 é apresentada uma ferramenta de avaliação de matrizes de projeto desenvolvida durante o trabalho de pesquisa. Por fim, no capítulo 8 são apresentadas as conclusões da dissertação e as perspectivas de trabalhos futuros.

2 Fundamentação Teórica

Neste Capítulo é apresentada a fundamentação teórica deste trabalho. Inicialmente é abordada a engenharia de sistemas, que é um tópico de discussão importante, uma vez que a abordagem proposta relaciona-se com a especificação de requisitos no âmbito de sistemas. Em seguida, apresenta-se uma visão geral da Engenharia de *Software* (ES), tratando especialmente dos processos e métodos de ES relacionados com a abordagem deste trabalho. A seguir é apresentada uma visão geral sobre a Engenharia de Requisitos (ER). Nesta seção são também discutidos alguns métodos de ER propostos na literatura. Além disso, é realizada são discutidos os principais termos utilizados nesta abordagem, como Problemas, Necessidades e Requisitos. Esta discussão tem como objetivo dar fundamentação teórica aos termos utilizados na abordagem proposta. Finalmente, é realizada uma breve discussão sobre os principais problemas e dificuldades em projeto de *software*.

2.1 Engenharia de Sistemas

Esta seção apresenta uma breve discussão sobre a Engenharia de Sistemas (ESis). A discussão começa pela apresentação da visão geral de ESis, sua definição e principais conceitos. A seguir é apresentada a linguagem de modelagem de sistemas baseada em UML, denominada SysML, que tem alguns de seus conceitos aplicados na abordagem proposta.

2.1.1 Visão Geral de Engenharia de Sistemas

O conceito de Engenharia de Sistemas (ESis) surgiu no final da década de 1950 (WEILKIENS, 2007). Naquela época, com as evoluções tecnológicas em andamento, passou-se a desenvolver sistemas cada vez mais complexos. O desenvolvimento de sistemas começou a superar todas as dificuldades encontradas até então. A corrida espacial e militar, impulsionada pela guerra fria, trouxe consigo projetos de sistemas extremamente complexos que precisavam estar prontos o mais

rapidamente possível. Esta pressão enorme fez com que diversos métodos de engenharia de sistemas fossem desenvolvidos para atender às necessidades destes projetos.

Do ponto de vista de engenharia, um sistema pode ser definido como um artefato que consiste de componentes ou blocos. Estes blocos possuem um objetivo comum que não pode ser atingido por esses elementos de forma individual (WEILKIENS, 2007). Neste contexto, os blocos podem ser programas de computador, circuitos eletrônicos, partes mecânicas, pessoas ou qualquer outro elemento (INCOSE, 2004). Estes elementos produzem resultados que colaboram para alcançar os objetivos do sistema. Os resultados correspondem a propriedades, características, funções, comportamentos ou características de desempenho do sistema (NASA STI, Program, 2007).

Esta definição de sistemas é bastante ampla e pode ser aplicada, por exemplo, no sistema de fornecimento de energia elétrica de um país ou em um automóvel. O sistema de abastecimento de energia de um país, por exemplo, é composto por usinas, redes de transmissão, subestações, transformadores, entre outros, e tem como objetivo fornecer energia elétrica com qualidade e eficiência. Por sua vez, o automóvel é composto por motor, transmissão, suspensão e rodas, entre outros elementos que colaboram para o objetivo do sistema que é transporte de pessoas com conforto e segurança.

A palavra Engenharia, por sua vez, também tem um significado que, em geral, define-se como uma disciplina que utiliza métodos e ferramentas de maneira estruturada para desenvolver produtos (WEILKIENS, 2007), como estruturas, máquinas, aparelhos ou sistemas. Desta forma, o termo Engenharia de Sistemas descreve métodos usados para desenvolver sistemas. Em outras palavras, ESis é uma disciplina integrativa em que as contribuições de diversas engenharias, como mecânica, elétrica, eletrônica, computação, e outras, são reunidas para produzir um resultado coerente (NASA STI, Program, 2007). Um resultado é considerado coerente se o relacionamento entre as partes colaborar para que os requisitos sejam atendidos, respeitando as restrições impostas ao sistema.

De uma maneira mais ampla, a ESis é uma abordagem disciplinada para projetar, construir, gerenciar e operar um sistema. Ela envolve todas as etapas de desenvolvimento de um sistema, desde a concepção até desenvolvimento, utilização e substituição do sistema. Assim, a ESis se concentra na definição e documentação

de requisitos no início do desenvolvimento. Durante o processo de desenvolvimento, ela se preocupa com a elaboração do *design* do sistema e a verificação de que os requisitos estão sendo atendidos. Além disso, leva em conta a operação, teste, criação, custo, planejamento, treinamento e suporte do produto (NASA STI, Program, 2007). Finalmente, observa aspectos econômicos e técnicos para desenvolver um sistema que atenda as necessidades dos usuários (INCOSE, 2004).

As disciplinas envolvidas nas etapas de desenvolvimento de sistemas são tanto gerenciais quanto de desenvolvimento do projeto. A Figura 2 representa as disciplinas envolvidas na engenharia de sistemas. As atividades de gerenciamento incluem gerenciamento de projeto, gerenciamento de requisitos e gerenciamento de riscos. As atividades de projeto incluem análise de requisitos, definição de requisitos, projeto de sistema, validação e integração de sistema.



Figura 2 - Disciplinas de Engenharia de Sistemas
 Fonte: Adaptado de (WEILKIENS, 2007)

É possível notar que todas estas atividades possuem correspondentes nos processos de Engenharia de *Software* (ES). Isso se deve ao fato da ES também tratar do desenvolvimento de sistemas, mais especificamente sistemas de *software*. Portanto, a ES desenvolve processos, métodos e ferramentas específicos para o desenvolvimento de *software*. Enquanto a ESis, por sua vez, pretende integrar

diversas disciplinas como ES, eletrônica, mecânica e qualquer outra disciplina que possa estar envolvida na construção de um sistema.

Na próxima seção é apresentada uma linguagem de modelagem de sistemas que auxilia os engenheiros a especificar sistemas interdisciplinares. Ela possui elementos que podem ser utilizados na especificação, análise, verificação e validação de sistemas.

2.1.2 Modelagem de Sistemas com SysML

A UML é atualmente a linguagem de modelagem mais utilizada para o desenvolvimento de *software*. A UML é um padrão internacional especificado pela OMG (2010) e também é aceita como um padrão ISO (ISO/IEC, 2005). No entanto, com o passar do tempo, vem sendo notadas certas deficiências na linguagem. Uma dessas deficiências está na modelagem de requisitos não funcionais. Enquanto, na UML, os requisitos funcionais podem ser associados aos casos de uso não existe nenhum elemento na linguagem que descreva os requisitos não funcionais (WEILKIENS, 2007). Entretanto, como estes elementos são necessários na prática, surgiram diversas propostas para que fosse possível modelar requisitos com a UML, uma destas propostas é SysML (WEILKIENS, 2007).

A SysML surgiu por que, embora existisse uma série de iniciativas para padronizar processos de Engenharia de Sistemas (ESis) nenhuma linguagem de modelagem de sistemas havia sido proposta até então. A ausência de uma linguagem de modelagem de sistemas causava dificuldades especialmente na modelagem de sistemas interdisciplinares. Em 2001 o INCOSE (*International Council on Systems Engineering*) decidiu transformar a UML na linguagem padrão para a ESis. Por meio dos mecanismos de extensão da UML novos vocabulários de modelagem podem ser definidos, adaptando a UML para domínios específicos. A adaptação da UML para a ESis é chamada de OMG Systems Modeling Language (OMG SysML™) (HAUSE, 2006). A Figura 3 ilustra a relação entre a UML e a SysML.

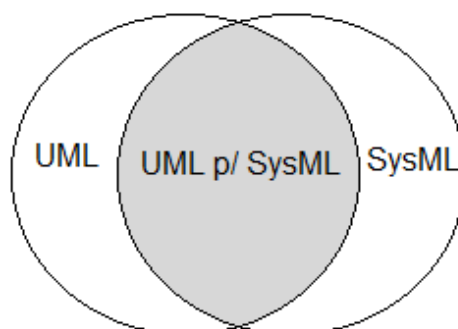


Figura 3 - Relação UML e SysML
Fonte: adaptado de (WEILKIENS, 2007)

A SysML fornece recursos para a especificação, análise, verificação e validação de um grande número de sistemas. Pretende-se, por meio dela, unificar uma série de linguagens de modelagem utilizadas pelos engenheiros de sistemas atualmente, da mesma forma que a UML unificou parte das linguagens de modelagem utilizadas na indústria de *Software*.

Como ilustra a Figura 3, a SysML aproveita uma parte dos diagramas e elementos da UML 2.0 e agrega novos elementos e diagramas por meio dos mecanismos de extensão definidos pela própria UML, que permitem a criação de perfis (*profiles*) (HAUSE, 2006). Como a SysML tem suas bases na UML, os engenheiros de sistema e engenheiros de *software* tem em mãos uma ferramenta que facilita a colaboração para a modelagem de sistemas complexos. É provável que no futuro esta linguagem venha a ser customizada para modelar aplicações de domínios específicos como automotivo, aeroespacial e telecomunicações.

A SysML possui diagramas que podem ser usados para especificar requisitos, comportamentos, estruturas e restrições do sistema. Esses são seus quatro pilares (WEILKIENS, 2007). Observando a Figura 4, que apresenta a hierarquia de diagramas propostos pela SysML, pode-se notar que a linguagem é fortemente baseada nos diagramas da UML. Somente dois novos diagramas foram criados, o Diagrama de Requisitos e o Diagrama Paramétrico. Por sua vez, o Diagrama de Atividades, o Diagrama de Definição de Blocos e o Diagrama de Blocos Internos são derivados de diagramas da UML. O diagrama de definição de blocos, por exemplo, é uma modificação do diagrama de classes da UML e é utilizado para descrever a hierarquia dos componentes do sistema.

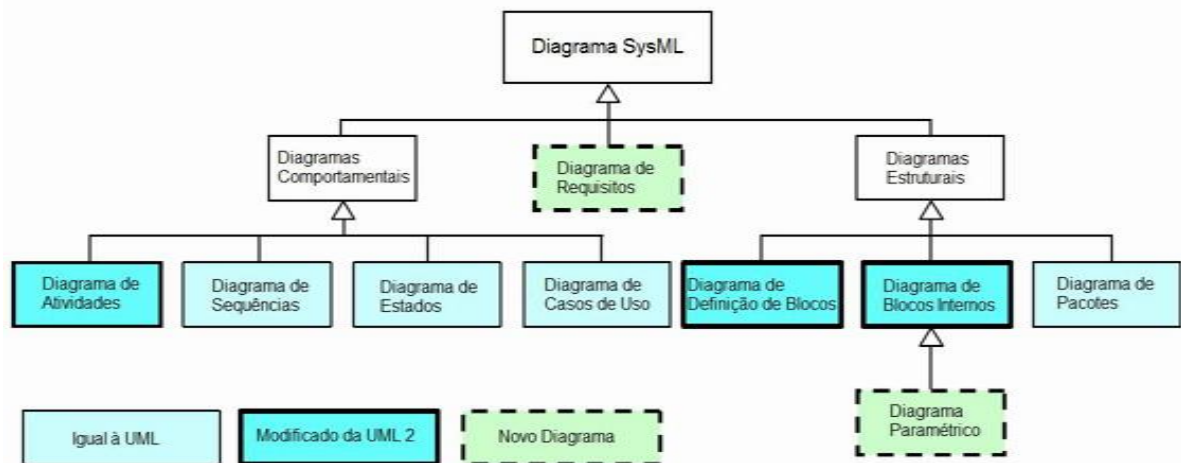


Figura 4 - Diagramas da SysML
 Fonte: (HAUSE, 2006)

Na SysML, a estrutura do sistema é representada por diagramas de definição de blocos e de blocos internos, ditos diagramas estruturais. Por sua vez, os diagramas comportamentais incluem diagrama de casos de uso, diagrama de atividades e diagrama de estados.

Particularmente, o diagrama de requisitos tem papel importante na abordagem proposta neste trabalho de dissertação. Isso se deve ao fato de permitir que requisitos funcionais e não funcionais sejam modelados na forma de diagrama, além de permitir a hierarquização de requisitos. Ele captura a hierarquia de requisitos e relacionamentos de derivação, verificação, satisfação e refinamento (SOARES & VRANCKEN, 2008). Esses relacionamentos permitem relacionar requisitos entre si e, também, com outros elementos como casos de uso, blocos e casos de teste, inclusive com fim de rastreabilidade.

A Figura 5 apresenta o exemplo de um diagrama de requisitos de um sistema de testes de equipamentos. O diagrama apresenta dois requisitos, “Teste de Equipamentos” e “Módulo de Relatórios de Testes”, que são decompostos em sub-requisitos. Os sub-requisitos de “Teste de Equipamentos” são “Teste de Impressoras” e “Teste de Placas”. O relacionamento entre os requisitos e sub-requisitos é de derivação, representado pelo estereótipo «*derive*». Os sub-requisitos por sua vez podem possuir ainda outros sub-requisitos. Como é o caso de “Teste de Placas” que tem como sub-requisitos “Teste de Interface Serial” e “Teste de Memória”.

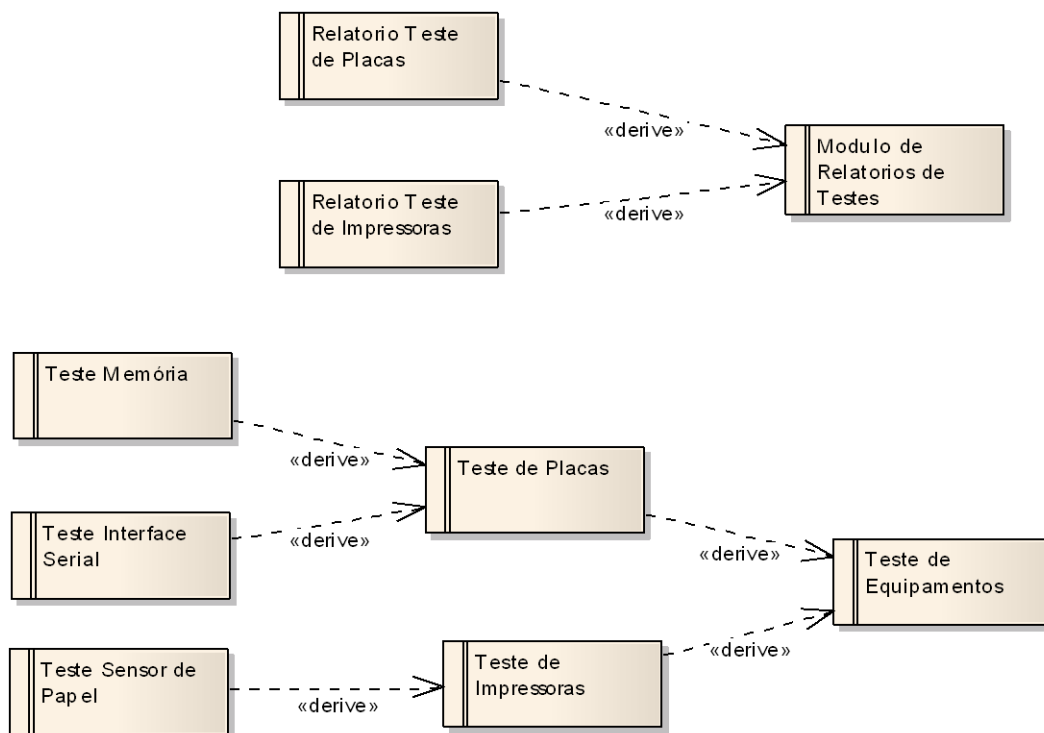


Figura 5 - Diagrama de Requisitos
Fonte: Autoria própria

Esta seção apresentou uma breve descrição da linguagem de modelagem de sistema (SysML). Na próxima seção são estudados conceitos e processos de Engenharia de *Software* também relevantes para este trabalho.

2.2 Engenharia de *Software*

Esta seção apresenta uma breve discussão sobre a Engenharia de *Software* (ES). A discussão começa com a apresentação da ES, sua definição e principais conceitos. A seguir é discutido o que é projeto de *software* e quais as principais questões relacionadas a esse assunto. O próximo tópico discutido é relativo a processos de *software*, no qual é apresentado o Processo Unificado que é fundamental neste trabalho por se propor a aplicação da abordagem proposta no contexto deste processo. Finalmente, apresenta-se a UML, linguagem de modelagem mais utilizada atualmente na especificação de projetos de *software* orientado a objetos.

2.2.1 Visão Geral da Engenharia de Software

A definição de engenharia de *software*, segundo o IEEE (1990), é a seguinte: “(1) aplicação de uma abordagem sistemática, disciplinada e quantitativa, para o desenvolvimento, operação e manutenção do *software*; (2) o estudo de abordagens como as de (1) (IEEE, 1990).”

Em termos mais simples, pode-se dizer que a Engenharia de *Software* é uma área de conhecimento que envolve a especificação, desenvolvimento e manutenção de *softwares* com objetivo de garantir a organização, a produtividade e a qualidade dos produtos e processos (PRESSMAN, 2006). No intuito de atender esses objetivos, existem numerosas práticas e métodos que visam apoiar os engenheiros de *software*. Pode-se citar: UML, Processo Unificado, CMMI, MPSBR e Processos ágeis.

Pressman (2006) representa a Engenharia de *Software* em camadas, conforme ilustrado na Figura 6. Segundo esta visão, a base que apoia a Engenharia de *Software* é o foco na qualidade. O processo é a segunda camada que forma o alicerce da engenharia de *software*, pois forma a base para o controle gerencial de projetos de *software*. Por sua vez, o processo estabelece o contexto no qual os métodos são aplicados, os produtos de trabalho são gerados, os marcos são estabelecidos, a qualidade é assegurada e as modificações são geridas. Os métodos fornecem a técnica de como construir o *software*. Eles envolvem questões como análise de requisitos, modelagem de projeto, construção de *software* e testes. São exemplos de métodos a Casa da Qualidade (HOQ), a Teoria Inventiva de Resolução de Problemas (TRIZ), etc. Finalmente, as ferramentas fornecem o apoio para a realização dos processos e métodos. Podem-se citar como exemplos de ferramentas de Engenharia de *software* *MS Visio*, *Enterprise Architect*, *MS Project*, entre outros.



Figura 6 - Engenharia de *Software* em Camadas
Fonte: Adaptado de (Pressman, 2006)

Todos os processos, métodos e ferramentas da ES têm como objetivo facilitar e tornar mais eficiente a elaboração de projetos de *software*, bem como garantir sua qualidade. Na próxima seção são discutidas as principais questões relativas a projeto de *software*. Em seguida são discutidos os processos e métodos de ES mais relevantes para esta pesquisa.

2.2.2 Projeto de *Software*

O ato de projetar está relacionado com a elaboração de um modelo ou plano com objetivo de orientar a realização de um projeto. Isso frequentemente é feito com base em métodos que orientam a criação deste modelo e na experiência prévia de quem elabora o projeto. Um projeto é elaborado a partir de uma necessidade que precisa ser atendida. Esta necessidade será satisfeita pelo produto resultante do projeto. Desta forma, o processo de projetar transforma a percepção de um problema em um objeto de projeto, ou seja, em um produto.

Quando o assunto é *software*, projetar significa determinar como os requisitos do cliente serão *implementados* na forma de estruturas de *software* (PRESSMAN, 2006). Ou seja, durante a elaboração do projeto, o conjunto dos requisitos é mapeado para um conjunto de estruturas que comporão o *software*. Este processo de mapeamento resulta na definição da estrutura interna do *software* que é chamada arquitetura. Arquitetura de *software* pode ser definida como a estrutura ou organização dos componentes do programa (módulos), a maneira pela qual estes componentes interagem entre eles e com o exterior, além da estrutura da informação que é usada por estes componentes (PRESSMAN, 2006).

A principal razão para se elaborar um projeto de *software* é desenvolver uma arquitetura que ajude a construir um produto de *software* de qualidade. De acordo com Pressman (2006), existem diretrizes para a qualidade de um projeto de *software*, entre elas estão:

1. Um projeto deve ser modular;
2. Um projeto deve conter representações distintas para dados, arquitetura, interfaces e componentes;

3. Um projeto deve levar a componentes que possuam características de independência funcional;
4. Um projeto deve levar a interfaces que reduzam a complexidade das conexões entre os componentes e o ambiente externo.

As diretrizes de qualidade citadas podem ser usadas para avaliar a qualidade de um projeto de *software*. Mas para alcançar essas diretrizes, é muito importante que o projetista seja disciplinado na aplicação de um processo de desenvolvimento que leve em conta estas diretrizes.

Analisando as diretrizes 1, 3 e 4, percebe-se que são diretamente relacionadas a dois conceitos bastante conhecidos em engenharia de *software*, Acoplamento e Coesão. De acordo com Stevens, Myers, & Constantine (1974), acoplamento é a medida de força da associação estabelecida por uma conexão entre dois módulos. Coesão, por sua vez, é definida como a medida da força da relação funcional dos elementos dentro de um módulo (PAGE-JONES, 1988). Em projeto de *software* busca-se sempre reduzir o acoplamento e aumentar a coesão com o objetivo de melhorar a qualidade do *software* de maneira geral, bem como aumentar sua manutenibilidade (PAGE-JONES, 1988).

No próximo capítulo é apresentada a Teoria de Projeto Axiomático e as razões pelas quais seus princípios colaboram para o atendimento das diretrizes apresentadas por Pressman (PRESSMAN, 2006). Princípios estes que se assemelham aos conceitos de acoplamento e coesão citados no parágrafo anterior. Outra questão relevante a respeito da arquitetura de *software* é que não basta elaborar uma arquitetura, é necessário ser capaz de avaliar se esta é a arquitetura adequada. Uma vantagem do projeto axiomático é justamente oferecer ferramentas que auxiliam os projetistas a realizar esta avaliação das arquiteturas de projeto que elaboram.

2.2.3 Processo de Software

Esta seção apresenta uma visão geral a respeito de processos de desenvolvimento de *software*. A seguir é discutido o Processo Unificado (PU) que é amplamente utilizado e conhecido por empresas de *software*. A abordagem de

especificação de requisitos proposta neste trabalho pode ser utilizada em um processo como este. Por esta razão, mais adiante é discutida a aplicação da abordagem nas fases do PU.

2.2.3.1 Visão Geral de Processo de Software

Um processo de desenvolvimento de *software* pode ser definido como um conjunto coerente de atividades, políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários para conceber, desenvolver, disponibilizar e manter produtos de *software* (FUGGETTA, 2000).

Pode-se dizer que o processo de *software* fornece conceitos abrangentes para organizar diferentes aspectos das atividades de desenvolvimento de *software* desde sua concepção até sua entrega ao cliente final. Para Pressman (2006), “processo de *software* é um arcabouço para as tarefas que são necessárias para construir *softwares* de alta qualidade”. Desta forma, o processo define a abordagem a ser adotada na elaboração de um projeto de *software*.

O objetivo principal dos processos é garantir o desenvolvimento de um produto que solucione os problemas do cliente e tenha qualidade. Além disso, deve auxiliar na manutenção de custos e prazos dentro dos patamares estimados. Em outras palavras, um processo de desenvolvimento estabelece a sistemática que organiza as atividades de desenvolvimento com vista aos resultados que devem ser produzidos (CHRISSIS, KONRAD, & SHRUM, 2007).

Existem diversos modelos de processos de *software* em uso no mercado que podem ser classificados de duas maneiras distintas: Processos de *Software* Definidos e Processos de *Software* Empíricos (SCHWABER, 2004).

Processos de *Software* Definidos são processos que repetidamente produzem resultados de qualidade aceitável (SCHWABER, 2004). É nesta classe que se enquadram o Processo Unificado e outros modelos como o Modelo em Cascada, Modelo Incremental, Modelo em Espiral e Modelo Concorrente (PRESSMAN, 2006). Também se inclui nesta categoria os modelos de processo como o CMMI, que guia a elaboração de processos definidos (CHRISSIS, KONRAD, & SHRUM, 2007).

Processos de *Software* Empírico são utilizados quando um processo definido não pode ser aplicado devido à complexidade de atividades intermediárias (SCHWABER, 2004). Desta forma, opta-se pela simplificação dos processos de desenvolvimento em favor da agilidade. Os modelos denominados Processos Ágeis, como *Extreme Programming* e *Scrum*, podem ser qualificados como Processos Empíricos (SCHWABER, 2004).

O Processo Unificado é um dos processos mais conhecidos e utilizados no desenvolvimento de *software*. Também é o processo utilizado na abordagem de Pimentel (2007), que inspirou o desenvolvimento desta pesquisa. Por esta razão este processo é apresentado na próxima seção.

2.2.3.2 Processo Unificado

O Processo Unificado de desenvolvimento de *software* é o conjunto de práticas que guiam o desenvolvimento de *software* para transformar requisitos do usuário em um sistema de *software*. Uma de suas características é possuir um ciclo de vida iterativo e incremental (JACOBSON, BOOCH, & RUMBAUGH, 1999). A cada iteração, um avanço do projeto é elaborado, desenvolvido, testado e integrado ao restante do projeto (PRESSMAN, 2006). Essa abordagem ajuda a lidar com a complexidade do desenvolvimento de sistemas maiores e torna o desenvolvimento flexível para admitir novos requisitos ou a mudança de requisitos.

Outra característica importante do processo unificado é ser guiado por casos de uso durante todo seu ciclo de vida. Por sua vez, as atividades de construção são baseadas na arquitetura do sistema (JACOBSON, BOOCH, & RUMBAUGH, 1999). Isso significa que o desenvolvimento consiste em construir o produto com base na arquitetura (PRESSMAN, 2006). O processo unificado reconhece a importância da comunicação com o cliente e do contato direto com o mesmo para a definição de suas necessidades. Os casos de uso são utilizados como meio de comunicação entre desenvolvedores e clientes. Uma vez que casos de uso devem ser simples de entender e devem servir como forma de verificar que as necessidades do cliente foram claramente entendidas.

A arquitetura global do processo unificado possui duas dimensões. A primeira dimensão, no eixo horizontal, representa o tempo ou fases do processo. A

segunda, no eixo vertical, representa os fluxos essenciais do projeto. Esta estrutura está representada na Figura 7.

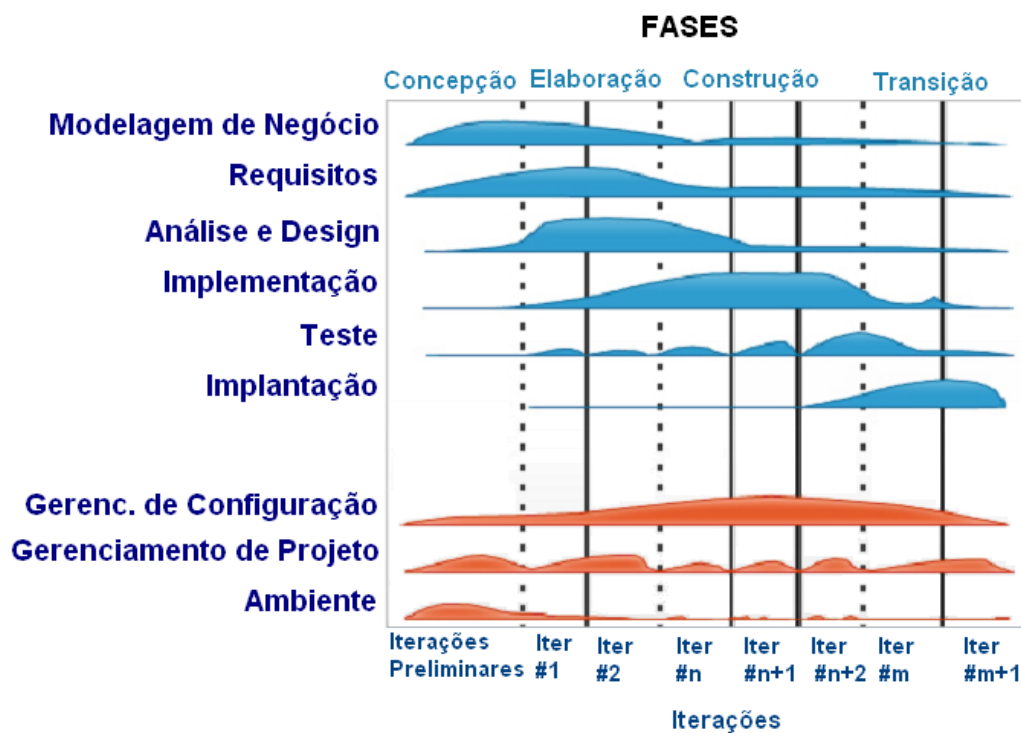


Figura 7 - Estrutura do Processo Unificado
 Fonte: Adaptado de (KRUCHTEN, 2003)

A primeira dimensão representa o aspecto mais dinâmico do processo, expresso por meio de fases e iterações (KRUCHTEN, 2003). As quatro fases do Processo Unificado são concepção, elaboração, construção e transição. A segunda dimensão representa os aspectos estáticos do processo como atividades, fluxos, artefatos e papéis de trabalhadores (KRUCHTEN, 2003). Esta segunda dimensão está dividida em nove disciplinas que incluem: modelagem de negócio, requisitos, análise e *design*, *implementação*, teste, implantação, gerencia de configuração, gerenciamento de projeto e ambiente.

Cada disciplina pode ter maior ou menor ênfase dependendo da fase em que o projeto se encontra. Por exemplo, a disciplina de requisitos tem maior ênfase nas fases iniciais do projeto, quando são definidos os requisitos e modelados os casos de uso. No restante do projeto os requisitos têm menos destaque por estarem estabilizados e necessitarem apenas de acompanhamento e verificação da sua *implementação*. Por sua vez, a disciplina de implantação, por sua vez, tem pouca ou quase nenhuma ênfase em quase todas as fases do projeto, sendo mais necessária

a partir das etapas finais da fase de construção e tendo sua maior importância na fase final, de transição, em que é entregue para o cliente.

De acordo com (KRUCHTEN, 2003), o Processo Unificado reúne um conjunto de melhores práticas de *software*, quem incluem: desenvolvimento iterativo, gerenciamento de requisitos, desenvolvimento centrado na arquitetura, modelagem baseada em UML, qualidade de processo e produto e gerenciamento de mudanças. Este conjunto de práticas certamente agrega um valor significativo ao processo unificado como um todo, contribuindo em diversos aspectos para a melhoria dos resultados do projeto.

Conforme foi discutido, o PU é centrado na arquitetura e a modelagem desta arquitetura é baseada na UML. Da mesma maneira, a abordagem proposta nesta dissertação tem uma relação estreita com esta linguagem. Por esta razão, na próxima seção é discutida a modelagem de sistemas de *software* com UML.

2.2.4 Modelagem de Software com a UML

Esta seção discute o papel da UML na modelagem de *software*, especialmente *software* orientado a objetos. É apresentado um breve histórico da linguagem e, em seguida, seus principais conceitos e diagramas.

2.2.4.1 Visão Geral da UML

A UML (*Unified Modeling Language*) é uma linguagem padrão para a elaboração de projetos de *software*. Foi desenvolvida em conjunto por Booch, Rumbaugh e Jacobson na empresa Rational em meados década de 90 (BOOCH, RUMBAUGH, & JACOBSON, 2006). O grupo juntou elementos de linguagens próprias dos autores e de vários outros métodos usados na época com objetivo de criar uma linguagem que atendesse as necessidades de projetos de *software* orientados a objetos. A primeira versão da UML (0.9) foi lançada em junho de 1996.

A OMG (*Object Management Group*) adotou a especificação da UML 1.1 em novembro de 1997 e, desde então, a linguagem passou por diversas revisões (OMG, 2010). A UML se tornou a linguagem de modelagem mais utilizada em projetos de sistemas de *software* orientados a objetos. Tem sido aplicada em uma ampla

diversidade de domínios, desde aplicações de saúde e finanças até aplicações de comércio eletrônico e aeroespaciais (OMG, 2010).

De acordo com a definição formulada pela OMG, “a UML é uma linguagem destinada a visualizar, especificar, construir e documentar os artefatos de um sistema de *software*” (OMG, 2010). Em resumo, é uma linguagem de modelagem visual, cujo objetivo é fornecer um conjunto de regras e modelos que tornem simples a elaboração de um projeto de sistema. Assim, torna-se possível compreender o sistema visualmente e, conseqüentemente, desenvolvê-lo dentro das especificações do cliente.

A UML por si só é apenas uma linguagem de modelagem, portanto é apenas uma ferramenta empregada em um processo de desenvolvimento de *software* (BOOCH, RUMBAUGH, & JACOBSON, 2006). Entretanto, ela é independente do processo de desenvolvimento de *software*, podendo ser utilizada em conjunto com processos ágeis, processo unificado ou processos customizados.

2.2.4.2 Conceitos da UML

A UML é uma linguagem desenvolvida especialmente para a modelagem de sistemas orientados a objetos. Por esta razão, boa parte de seus elementos representa conceitos do paradigma orientado a objetos (POO). Estes conceitos são discutidos nesta seção.

As *classes* são os blocos de construção mais importantes de qualquer sistema orientado a objetos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos tipos de atributos, operações e relacionamentos (BOOCH, RUMBAUGH, & JACOBSON, 2006). As classes especificam o estado e o comportamento dos objetos por meio de atributos e métodos, respectivamente.

Um *objeto* é uma instância de uma classe, à qual o conjunto de métodos definidos pela classe poderá ser aplicado e que possui um conjunto de atributos cujos valores representam seu estado (BOOCH, RUMBAUGH, & JACOBSON, 2006).

Poucos objetos em POO permanecem sozinhos, a maior parte das classes preveem colaborações entre seus objetos. Isto ocorre porque a comunicação entre os objetos é essencial para que estes executem suas tarefas dentro do sistema. De

acordo com Booch, Jacobson e Rumbaugh (2006), em orientação a objetos existem três tipos de relacionamento entre classes. Esses relacionamentos são agregação, associação e herança.

Outro conceito muito importante da UML são os casos de uso, que têm a função de representar os requisitos funcionais do sistema na linguagem (BOOCH, RUMBAUGH, & JACOBSON, 2006). O termo “caso de uso” é definido como a especificação de um conjunto de ações que representa uma utilização completa do sistema (PIMENTEL, 2007). Os atores representam as entidades externas que interagem com os casos de uso.

Os elementos básicos da UML são utilizados na elaboração de várias visões do sistema sob a forma de diagramas. Diagramas são formas de representação gráfica que, geralmente, são desenhadas para permitir a visualização de um sistema em diferentes aspectos. Um diagrama representa uma visão parcial de elementos que compõem o sistema e pode representá-lo em aspectos estáticos ou dinâmicos (BOOCH, RUMBAUGH, & JACOBSON, 2006). Os principais diagramas estáticos são: diagrama de casos de uso, diagrama de classes, diagrama de objetos, diagrama de componentes e diagrama de implantação. Entre os principais diagramas dinâmicos estão: diagrama de sequência, diagrama de atividades, diagrama de comunicações e diagrama de estados.

2.2.4.3 Perfis UML

A UML fornece um conjunto de conceitos de modelagem e notações que foram desenvolvidos para satisfazer as necessidades de projetos de *software*. Entretanto, em alguns casos são necessárias características adicionais de modelagem além daquelas definidas pela UML. Estas características pertencem a domínios específicos de aplicação como aeronáutica, finanças e saúde. A UML permite a modelagem dessas necessidades específicas por meio dos seus mecanismos de extensão, que permitem a adição de novos elementos de modelagem ao metamodelo da UML. Um conjunto desses mecanismos de extensão, agrupados dentro de um pacote UML marcado com o estereótipo «*profile*», forma um Perfil UML (FUENTES & VALLENCILLO, 2004). Um perfil é definido utilizando

estereótipos, *tags* ou restrições que são aplicados a elementos específicos da UML, como classes, atributos, operações e atividades .

Existem, atualmente, muitos perfis públicos definidos para diferentes áreas de aplicação. Um exemplo de perfil UML, citado anteriormente neste trabalho, é a SysML, um perfil padronizado pela OMG (Object Management Group) que é utilizado para aplicações de engenharia de sistemas (OMG, 2010). Outro exemplo de perfil é o MARTE (*Modeling e Analysis of Real Time and Embedded Systems*), utilizado na modelagem de aplicações embarcadas em tempo real (OMG, 2007).

A abordagem proposta neste trabalho mescla alguns elementos de UML, SysML e UML Profile. O ideal seria criar um perfil específico para a abordagem, o que não ocorreu por não ser o foco deste trabalho. Talvez a definição deste perfil possa vir a ser objeto de trabalhos futuros.

2.3 Engenharia de Requisitos

Esta seção apresenta uma visão geral da Engenharia de Requisitos (ER) e também alguns trabalhos importantes nesta área. Alguns destes trabalhos focam nas técnicas de especificação de requisitos, como Quadros de Problemas e Engenharia de Requisitos Orientadas a Objetivos. Outros focam tanto nas técnicas quanto na gestão de requisitos, como é o caso do Processo Unificado e a Abordagem de Casos de Uso apresentada em (LEFFINGWELL & WIDRIG, 2003).

2.3.1 Visão Geral de Engenharia de Requisitos

A ER é definida por Nuseibeh and Easterbrook (2000) como a área da Engenharia de *Software* (ES) que se preocupa com os objetivos reais das funcionalidades do *software*. Assim, a Engenharia de Requisitos age como uma ponte entre as reais necessidades de usuários, clientes e outras pessoas afetadas pelo sistema e a solução final.

Sabe-se que é extremamente importante compreender a real necessidade do cliente antes de iniciar o desenvolvimento de um sistema. Por esta razão, durante a fase de especificação, o conjunto das necessidades do cliente é mapeado para um conjunto de requisitos que definirão o que o *software* deverá fazer. Estes requisitos

futuramente serão mapeados para o conjunto de estruturas que comporão o *software*.

No meio de desenvolvimento de *software*, a ER é definida como uma área da ES. Por outro lado, sabe-se que um *software* costuma ter pouca utilidade se isolado do sistema ao qual pertence. Por este motivo, em uma visão mais abrangente, pode-se considerar a ER uma área da Engenharia de Sistemas.

A base que sustenta a ER são os processos, que determinam as atividades que devem ser realizadas e as técnicas, que focam as abordagens a serem utilizadas pelos analistas para identificar os requisitos. A seguir apresenta-se o processo de ER mais frequentemente aceito nos processos de *software*, como no Processo Unificado. Além disso, são discutidas duas técnicas de especificação de requisitos, Engenharia de Requisitos Orientada a Objetivos e Quadros de Problemas.

2.3.2 Processo de Engenharia de Requisitos

A ER de sistemas novos pode ser dividida em algumas atividades principais: analisar do problema, entender as necessidades dos clientes e outros envolvidos, definir o sistema, gerenciar o escopo e refinar a definição do sistema (KRUCHTEN, 2003). A Figura 8 apresenta a dinâmica destas atividades na visão do Processo Unificado.

A análise do problema é o processo de entender os problemas e as necessidades do usuário (LEFFINGWELL & WIDRIG, 2003). Esta atividade é realizada em cinco passos: (1) definir o problema e obter o aceite do cliente sobre esta definição; (2) estudar das causas dos problemas; (3) identificar os usuários e demais envolvidos; (4) definir a solução dos limites do sistema; (5) e identificar as restrições que são impostas à solução. Além disso a análise do problema inclui a modelagem do negócio que tem por objetivo descrever os processos do negócio, ou seja o contexto no qual o sistema estará inserido.

Entender as necessidades do cliente é certamente a tarefa mais desafiadora da ER, pois está fortemente relacionada à comunicação das necessidades que precisam ser atendidas pelo sistema. O cliente tem o conhecimento do domínio do problema e, portanto, sabe o que é necessário para atender suas necessidades.

Porém, é comum que o cliente tenha dificuldade em expressar isso verbalmente. Por outro lado, os analistas de requisitos podem ter dificuldades em entender o ponto de vista do cliente e seu entendimento de como o sistema pode resolver seus problemas. Entre as técnicas utilizadas para entender as necessidades e especificar requisitos pode-se citar questionários, análise de documentos e processos organizacionais, estudo de padrões, prototipagem, reuniões, *brainstorms*, entre outros (NUSEIBEH & EASTERBROOK, 2000).

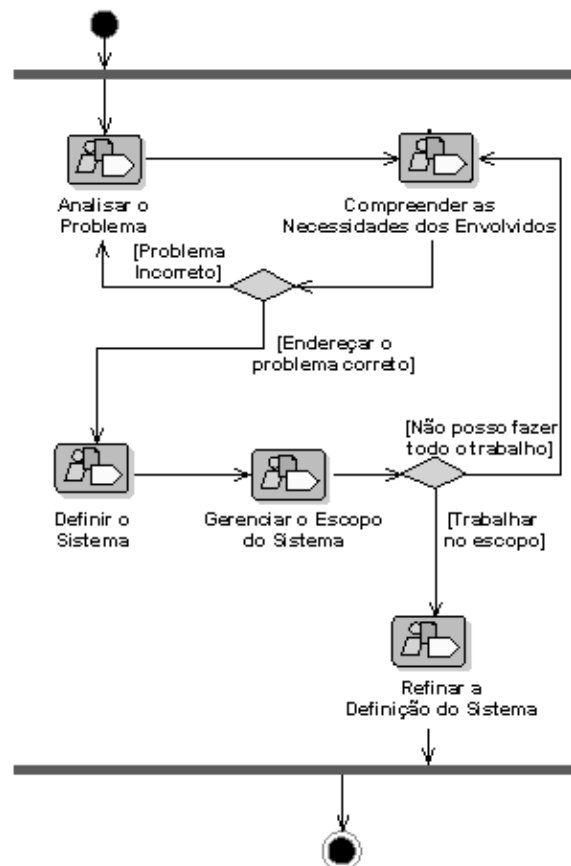


Figura 8 - Fluxo de Requisitos do Processo Unificado
 Fonte: Adaptado de (KRUCHTEN, 2003)

A definição do sistema consiste em organizar as informações obtidas a respeito dos requisitos, elaborar o documento de visão do sistema e descrever seus casos de uso. Os casos de uso incorporam a maior parte dos requisitos para dentro do sistema. Além disso, eles têm a vantagem de seguir um formato padronizado. No entanto, os casos de uso não modelam as restrições e requisitos não funcionais do sistema. Por esta razão é necessário elaborar uma documentação suplementar para incluir estes elementos na definição do sistema (LEFFINGWELL & WIDRIG, 2003).

O gerenciamento dos requisitos não é uma atividade isolada no início da especificação do sistema, uma vez que corre em paralelo com o processo de desenvolvimento do sistema. Essa atividade implica em gerenciar o escopo para garantir que todos os requisitos sejam *implementados*. Além disso, significa controlar alterações nos requisitos com objetivo de evitar alterações de escopo que impactarão nos custos, prazos e na qualidade do projeto.

Refinar a definição do sistema significa realizar um detalhamento dos requisitos identificados na etapa de estudo das necessidades. Este refinamento permite capturar o maior número de detalhes possível a respeito de cada requisito ou caso de uso. Desta forma, ao desenvolver a arquitetura, os analistas de sistemas terão os subsídios necessários para fazer a especificação do sistema.

O processo de requisitos descrito nesta seção envolve todas as etapas fundamentais da ER. No entanto, as técnicas utilizadas na análise do problema e no entendimento das necessidades parecem não ter o resultado esperado, uma vez que os números mostram que uma grande parte dos projetos falha por problemas com os requisitos (DOMINGUEZ, 2009). Talvez isso se deva ao fato de que estas técnicas não seguem um método bem definido e dependem basicamente da comunicação verbal entre usuários e desenvolvedores. Por esta razão, têm sido desenvolvidas técnicas para especificação de requisitos, como as discutidas a seguir.

2.3.3 Abordagem de Especificação de Requisitos Orientada a Objetivos

A abordagem de Engenharia de Requisitos Orientada a Objetivos (EROO), do termo em inglês *Goal-Oriented Requirements Engineering*, é também denominada KAOS. Este é um método de especificação de requisitos, que se dedica à análise de objetivos, requisitos e cenários do sistema (DARDENNE, van LAMSWEERDE, & FICKAS, 1993). Kaos também pretende suportar o processo de requisitos completo, desde a elaboração dos objetivos de alto nível até os requisitos e objetos do sistema. Além disso, oferece linguagem de modelagem por meio dos elementos que a compõem (van LAMSWEERDE & WILLEMET, 1998).

Esta abordagem se preocupa inicialmente em identificar os objetivos pretendidos para o sistema que será elaborado. Estes objetivos podem ser

decompostos em sub-objetivos que, por sua vez, também podem ser decompostos. Conforme os objetivos são decompostos, começam a surgir novos elementos, como requisitos, agentes, eventos, operações, entre outros (RESPECT-IT, 2007).

O processo de identificação dos objetivos pode não ser um processo de cima para baixo, mas também de baixo para cima na hierarquia dos objetivos. Isso se justifica pelo fato de que os usuários podem definir objetivos intermediários (RESPECT-IT, 2007). Os objetivos específicos podem ser identificados pelos analistas perguntando “Como iremos atender esse objetivo?”, enquanto os objetivos mais gerais podem ser identificados perguntando “Por que queremos fazer isso?” (van LAMSWEERDE, DARIMONT, & MASSONET, 1995).

KAOS apresenta algumas vantagens como a identificação de requisitos robustos e rastreabilidade vertical (van LAMSWEERDE, 2004), devido ao processo de hierarquização dos elementos. Além disso, oferece uma linguagem para comunicação entre os desenvolvedores e os usuários, que estão inicialmente mais interessados nos objetivos e estratégias da empresa (LI, 2007).

Por outro lado, este método foca primeiramente os objetivos e na forma como são decompostos (LI, 2007). Desta forma, deixa de lado o contexto em que o sistema será utilizado. Assim, algumas vezes os objetivos não serão decompostos em um conjunto adequado de “sub-objetivos”. Esses “sub-objetivos” podem levar à definição de um sistema que não atende às necessidades do usuário ou que é mais difícil de ser satisfeito pelo *software*.

2.3.4 Quadros de Problema

Na abordagem de Quadros de Problema (QP) (Jackson, 2001) o autor afirma que é mais eficiente iniciar o processo de desenvolvimento com uma tarefa de decomposição do problema antes de passar para o projeto de uma solução. Neste processo, o problema é decomposto um conjunto de subproblemas. Cada subproblema terá, em fases posteriores, um requisito e uma *implementação* correspondente (Jackson, 2001).

A decomposição dos problemas em subproblemas oferece uma simplificação dos mesmos, de maneira que sejam mais fáceis de resolver. É importante ressaltar que a decomposição dos problemas é paralela, ou seja, os

subproblemas não são partes uns dos outros e suas soluções ocorrem paralelamente (JACKSON, 1999). No entanto, caso os subproblemas sejam decompostos novamente, existirá uma hierarquização da relação entre os novos subproblemas com aqueles que os deram origem. O resultado da decomposição deve ser um conjunto de quadros de problemas pré-estabelecidos.

De acordo com Hall et al. (2002), o desenvolvimento de *software* deveria aproveitar a sinergia que existe entre o domínio do problema e domínio da solução com objetivo de permitir aos desenvolvedores explorar requisitos e oportunidades de projeto. Esta ideia parece muito promissora e se destaca em relação à maioria das abordagens, que não se preocupam em aprofundar a análise dos problemas do cliente. A Figura 9 ilustra o modelo de desenvolvimento das pirâmides gêmeas desenvolvido para ilustrar a natureza interativa do processo de desenvolvimento.

No modelo das pirâmides gêmeas tanto estruturas do problema como da solução são detalhadas e enriquecidas. Este processo lembra bastante os processos em *zig-zag* apresentado no próximo capítulo (SUH, 2001). Esta semelhança abre a possibilidade de se unir o conceito de decomposição do problema de Jackson, com o processo em zig-zag e o mapeamento de domínios de Suh.

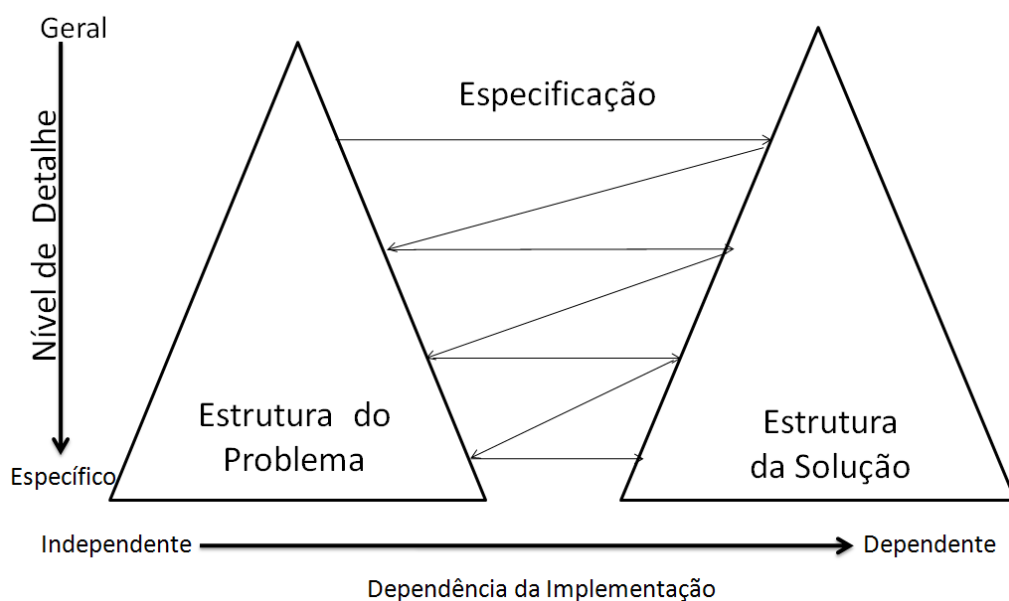


Figura 9 - Modelo das Pirâmides Gêmeas

Fonte: Adaptado de (HALL, JACKSON, LANEY, & NUSEIBEH, 2002)

Apesar de defender o uso de problemas e seu detalhamento como uma forma de identificar os requisitos do cliente, os trabalhos sobre esta abordagem não conseguem explicar de que maneira clara como seria o processo de composição da solução a partir dos quadros de problemas, o que dificulta sua aplicação comercial (LI, 2007).

Nesta seção foi apresentada a última técnica de especificação de requisitos discutida neste trabalho, repassando boa parte dos assuntos referentes à ER. Na próxima seção é apresentado um modelo de maturidade de requisitos, o que encerra o assunto e ER nesta fundamentação teórica.

2.3.5 Modelo de Maturidade de Requisitos

O conceito Maturidade de Requisitos é relativamente novo na Engenharia de Requisitos e também pouco difundido. Os trabalhos mais antigos a tratar do assunto datam de 2003 (HEUMANN, 2003). É possível que seja esta a razão de não ser amplamente conhecida, embora seja importante lembrar que as empresas que implantam modelos de maturidade como o CMMI precisam desenvolver processos de gestão de requisitos (CHRISISS, KONRAD, & SHRUM, 2007).

Em seu trabalho Heumann se refere ao modelo como *Requirements Management Maturity* (RMM) e define maturidade, no contexto de negócios, como a capacidade de tomar decisões baseado em uma compreensão clara do custo benefício de fazer uma escolha em detrimento de outra (HEUMANN, 2003). Ele estabeleceu cinco níveis para seu modelo de maturidade, que são: (1) Escrito; (2) Organizado; (3) Estruturado; (4) Rastreado e (5) Integrado.

Mais recentemente, uma empresa de consultoria chamada IAG apresentou uma nova proposta de modelo de maturidade de requisitos, o *IAG Requirements Maturity Model* (IRMM). O modelo possui duas dimensões, os Níveis de Maturidade e as Áreas de Capacidade.

Os níveis de maturidade definidos pela IAG seguem um padrão em estágios, similar àqueles utilizados na indústria como por exemplo no CMMI. A organização progride nos níveis conforme alcança os objetivos estabelecidos (ELLIS, 2009). Cada nível constrói uma base para os níveis seguintes. Os níveis de maturidade são cinco: (1) Executado; (2) Definido; (3) *Implementado*; (4) Institucionalizado e (5) Em

contínua otimização. A Figura 10 ilustra o modelo de níveis de maturidade em estágios.

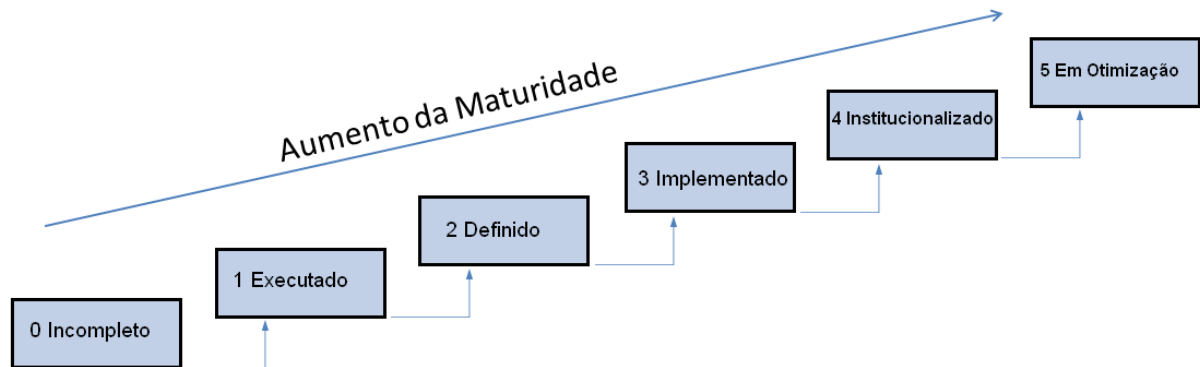


Figura 10 - Níveis de Maturidade de Requisitos
 Fonte: Adaptado de (ELLIS, 2009)

Por sua vez, as áreas de capacidade são disciplinas importantes para a especificação e gestão de requisitos e incluem: Processo, Práticas e Técnicas, Tecnologia, Competência Pessoal, Resultados e Organização. Processo se refere à definição, uso e gestão dos requisitos. As práticas e técnicas estão relacionadas à definição de como os analistas vão executar o trabalho, a eficiência e a eficácia dessas atividades. Tecnologia se refere ao fornecimento, uso e integração de ferramentas de *software* no contexto das práticas de requisitos. Competência pessoal preocupa-se com o nível de conhecimento, as habilidades e as capacidades dos colaboradores envolvidos na definição e gestão de requisitos. Resultados refere-se à definição, produção e uso de produtos de trabalho como saída do processo de requisitos. Organização preocupa-se com o modelo organizacional e serviços prestados às partes interessadas, o fornecimento e gestão de recursos na prestação desses serviços.

Estatísticas levantadas no trabalho de (ELLIS, 2009) são decorrentes da pesquisa Referencial de Análise de Negócios (*Business Analysis Benchmark*) realizada com 550 companhias escolhidas pela empresa de consultoria IAG. Para participar da pesquisa as empresas precisavam gastar acima de um milhão de dólares por ano em projetos. Os resultados desta pesquisa demonstram o impacto dos modelos de maturidade no desempenho de projetos, conforme pode ser observado na Figura 11. O que se percebe é que, a diferença nos percentuais de

projetos bem sucedidos nas quatro áreas analisadas é de aproximadamente quarenta por cento do primeiro para o quarto nível de maturidade.

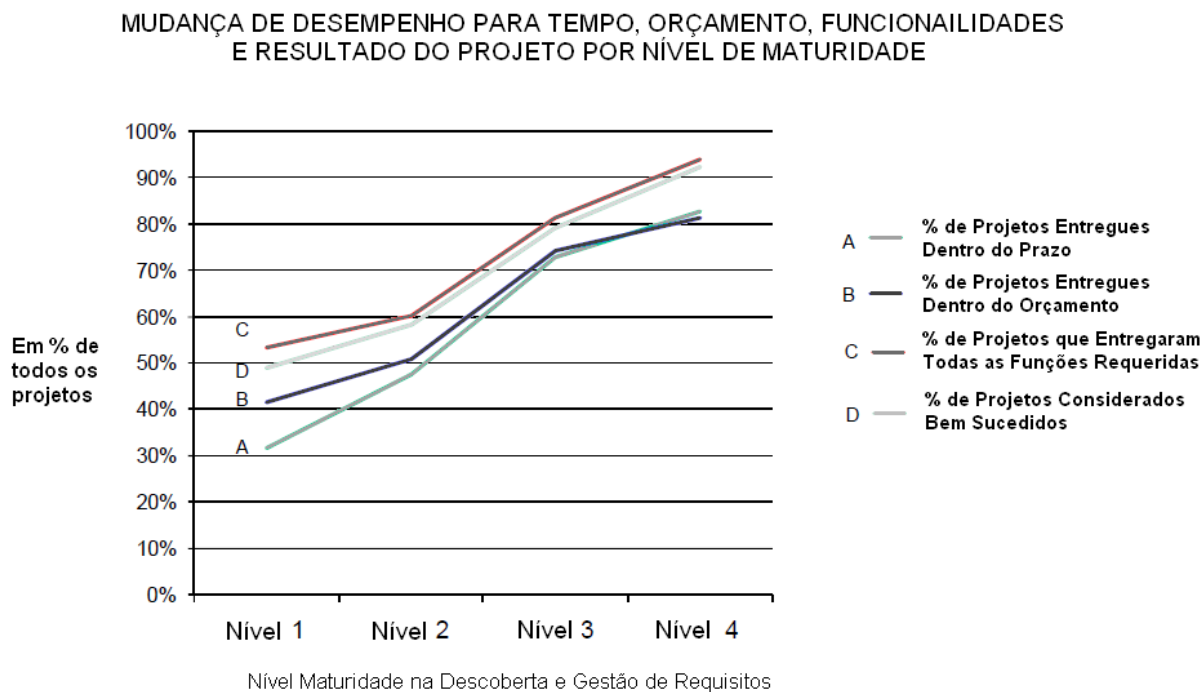


Figura 11 - Desempenho de Projetos por Nível de Maturidade de Requisitos
 Fonte: Adaptado de (ELLIS, 2009)

De maneira geral, a evolução da maturidade organizacional na definição e gestão de requisitos parece apontar para uma possível resposta para algumas questões não resolvidas até então. Por que, mesmo utilizando boas técnicas de especificação, algumas empresas não têm bons resultados. Assim, reunindo-se as técnicas com processos maduros é possível que a ER passe a ser uma disciplina mais consistente. O termo Requisito continua a ser discutido na próxima seção que estuda as definições dos principais termos envolvidos com a definição de requisitos.

2.4 Conceitos Relacionados à Especificação de Requisitos

Alguns termos como Necessidades do Cliente, Problemas e Características são comuns nos processos de Engenharia de Requisitos, principalmente nas fases iniciais do ciclo de vida de desenvolvimento. Sabe-se que é fundamental definir o problema que será resolvido. Da mesma forma que é importante compreender as

necessidades do cliente para identificar os requisitos de *software* adequados. Requisitos estes que, quando claramente definidos e compreendidos, levam ao sucesso do projeto. No entanto, estes termos não costumam ser bem compreendidos pelos analistas de requisitos. Sendo este trabalho dedicado ao estudo das questões relacionadas à especificação de requisitos, esta seção discute o significado destes termos e sua importância para a ER.

2.4.1 Necessidades Humanas

O objetivo de se estudar as necessidades humanas neste trabalho é compreender o grau de influência da questão na análise do problema, no estudo das necessidades do cliente e na definição de requisitos de sistema.

Em psicologia, Necessidade define um estado interno de insatisfação causado pela falta de algum item necessário ao bem estar da pessoa (CARVER & SCHEIER, *Perspectives on Personality*, 2000). As necessidades causam um estado de desequilíbrio interno e, por esta razão, motivam os comportamentos humanos.

Segundo Bergamini (1997), existem carências interiores não supridas que determinam um estado de desequilíbrio que gera desconforto negativo e ameaça a integridade do indivíduo. Assim, o indivíduo é motivado a suprir tais necessidades e restituir o equilíbrio interno satisfatoriamente.

Henry Murray foi um dos primeiros autores a estudar e classificar as necessidades humanas (NHs) (MURRAY, 1938). Segundo ele, existem dois tipos de necessidades: primárias e secundárias. As necessidades primárias possuem natureza biológica, como fome, sede, sono, entre outras. Por sua vez, as necessidades secundárias são aprendidas no decorrer da vida, de acordo com estruturas físicas, sociais e culturais do ambiente.

Murray desenvolveu uma lista de necessidades que, segundo ele, são comuns a todas as pessoas (MURRAY, 1938). O que diferencia as pessoas é a proporção em que cada uma destas necessidades é mais ou menos importante. De acordo com o autor, as necessidades são ligadas a algum destes temas: ambição, objetos inanimados, defesa do *status*, poder humano, afeição ou troca de informação.

Três décadas após a publicação do livro de Murray, Abraham Maslow propôs outra teoria das necessidades humanas (MASLOW, 1970). A Teoria de Maslow no campo das necessidades humanas continua até hoje a influenciar as áreas de Administração e *Marketing*. De acordo com esta teoria, as necessidades humanas formam uma hierarquia, como uma pirâmide, como a ilustrada na Figura 12.

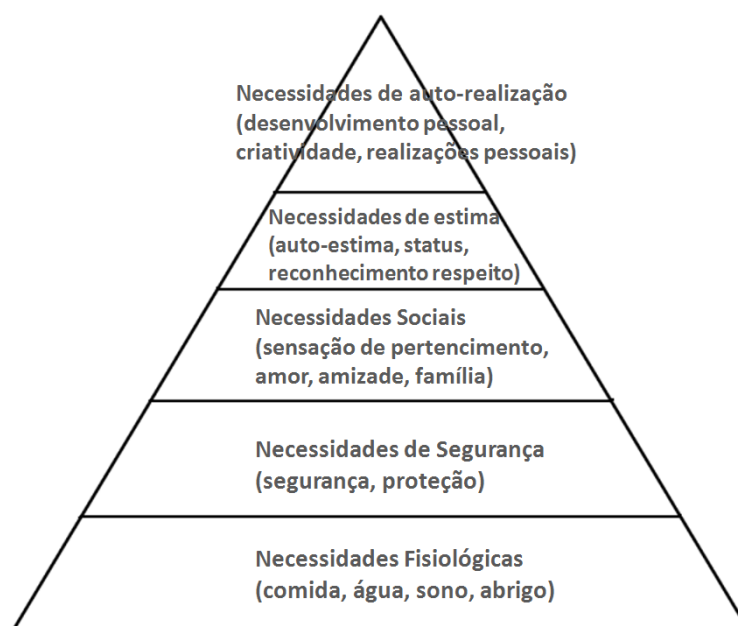


Figura 12 - Pirâmide das Necessidades de Maslow
Fonte: Adaptado de (MASLOW, 1970)

No nível mais baixo da pirâmide encontram-se as necessidades mais primitivas e prioritárias, por assim dizer, como comer, dormir e respirar. Estas necessidades são chamadas fisiológicas. No próximo nível estão necessidades também vitais, porém menos exigentes, e são chamadas necessidades de segurança. No terceiro nível as necessidades são relacionadas à vida social, incluindo as necessidades de amor e relacionamentos, como afeição, aceitação e interação com outras pessoas. No nível seguinte estão as necessidades de estima, na qual se enquadram o desejo de ser apreciado, ter poder ou ser bom em alguma coisa. No topo da pirâmide estão as necessidades de realização pessoal, ou seja, a necessidade de desenvolver as próprias potencialidades.

De acordo com Maslow (1970), as necessidades que estão na parte mais inferior da pirâmide são mais primitivas e urgentes do que as dos níveis mais elevados. Desta maneira, somente quando as necessidades mais básicas são

saciadas o próximo nível torna-se relevante para a pessoa. Além disso, Maslow diferencia as necessidades mais básicas das mais elevadas. As primeiras são necessidades defectivas, o que significa que se baseiam na falta e devem ser saciadas para evitar dor ou sofrimento. As outras, por sua vez, são necessidades de crescimento e buscam ser saciadas para se alcançar algo mais desejável.

Outra questão a ser apontada é que um dos princípios básicos do Marketing é que um produto deve ser criado com o objetivo de atender uma necessidade do mercado, ou seja, dos consumidores. No entanto, com o passar do tempo fica claro que nem sempre o que o mercado possui é uma necessidade, mas sim um desejo. Ao procurar por um produto ou serviço no intuito de atender a uma necessidade é comum que o cliente escolha aquele que, além de oferecer uma solução para seu problema, possui características agregadas que atendem seus desejos (HERNANDES, 2009). Normalmente, tudo o que se oferece adicionalmente à solução principal pode ser considerado como atendimento a desejos do cliente.

Nesta seção foram estudadas as necessidades que motivam o ser humano, que é o fornecedor dos requisitos de um sistema. Por outro lado, sabe-se que a maioria dos clientes de grandes projetos são, na verdade, organizações ou empresas. Sendo assim, a próxima seção estuda a questão das necessidades do ponto de vista das organizações.

2.4.2 Necessidades Organizacionais

Antes de se discutir as necessidades organizacionais, é importante definir o que é uma organização do ponto de vista deste trabalho de pesquisa. Pode-se definir uma organização como um grupo social que distribui tarefas que são executadas para a obtenção de um objetivo comum. Ainda, uma empresa pode ser particular, pública ou de economia mista.

De acordo com Moraes (2004):

“Organizações são instituições sociais e a ação desenvolvida por seus membros é dirigida por objetivos. São projetadas como sistemas de atividades e autoridade, deliberadamente estruturados e coordenados, elas atuam de maneira interativa com o meio ambiente que as cerca”.

Em outras palavras, organização pode ser definida como um conjunto organizado de meios para produzir e oferecer bens ou serviços, com o objetivo de

atender a alguma necessidade humana. É interessante notar que empresas atendem a necessidades e interesses de vários grupos, sendo eles: clientes, investidores, colaboradores e a sociedade (KOTLER & KELLER, 2006).

Segundo Chiavenato (2005), assim como as pessoas, as empresas ou organizações também possuem necessidades. Enquanto, por um lado, as pessoas nas empresas procuram por salários, segurança no cargo, crescimento na carreira e realização pessoal, por outro, as empresas necessitam de capital, equipamentos, lucratividade e capital humano, um dos fatores mais importantes.

Além disso, é importante perceber que são pessoas que tomam decisões de compra e estas são motivadas pelas próprias necessidades e percepções na tentativa de obter recompensas como salário, reconhecimento e sentimento de realização (KOTLER & KELLER, 2006). Por esta razão, nota-se que, dentro de uma organização, as pessoas não estão somente comprando produtos, mas sim comprando soluções para dois problemas: o problema econômico e estratégico da organização e seu próprio problema de alcançar realização pessoal e ser recompensando.

É pouco comum encontrar na literatura trabalhos dedicados à classificação das necessidades organizacionais (NOs) de forma similar aos estudos de (MURRAY, 1938) e (MASLOW, 1970) sobre as necessidades humanas. Assim, parece relevante elaborar uma classificação das NOs

Uma sugestão de necessidades organizacionais comuns nas empresas incluiria: recursos financeiros, recursos humanos, clientes, produção, estrutura física, obrigações, estabilidade financeira, posicionamento no mercado, planejamento, controle de processos, reconhecimento da marca, com relacionamento com clientes e mercado, ter credibilidade, possuir diferenciais, respeitar seus valores, gerar retorno financeiro, gerar retorno social, crescer e atingir seus objetivos. Com base nesta lista buscou-se classificar as necessidades das empresas por níveis, seguindo o modelo das necessidades humanas proposto por (MASLOW, 1970). Essa pirâmide sugere que as NOs poderiam ser classificadas da mesma maneira que as humanas, uma vez que organizações são concebidas e formadas por pessoas. O resultado desta classificação é apresentado na Figura 13.



Figura 13 - Hierarquia das Necessidades Organizacionais
Fonte: Autoria própria

Conforme estudado até aqui quando as necessidades organizacionais (ou humanas) estão em desequilíbrio causam incômodo, o que se classifica como “problema”. O termo problema é o assunto da próxima seção.

2.4.3 Problema

De maneira geral, o termo problema pode ser definido como um obstáculo, impedimento, dificuldade, desafio ou qualquer situação que leva ao desejo de uma resolução. De acordo com o dicionário Cambridge (2011), “Problema é algo que causa dificuldade ou com o qual é difícil de lidar.” Portanto, pode-se dizer que, do ponto de vista do ser humano comum, problema é tudo aquilo que lhe causa desconforto físico ou tensão psicológica. É possível também observar que, de maneira geral, os problemas surgem a partir do desequilíbrio de uma necessidade e também do medo de não ter alguma dessas necessidades satisfeitas.

Na Engenharia de *Software*, um problema costuma ser visto como uma dificuldade de um usuário que será resolvida por meio do sistema que será desenvolvido. Nas fases iniciais do ciclo de vida de desenvolvimento de *software* é bastante frequente a utilização de termos como Análise do Problema, Domínio do Problema ou, simplesmente, Problema. Menos frequente, no entanto, é encontrar

técnicas que auxiliem os analistas a trabalhar os problemas do cliente de forma eficiente.

A resolução de um problema é definida como a solução do mesmo. A busca pela solução de um problema é chamada de Resolução de Problemas (do termo em inglês *Problem Solving*). As quatro áreas mais importantes da Resolução de problemas são: Identificação do problema, Identificação das causas do problema, Identificação das consequências do problema e a Proposição de soluções.

Com frequência, as causas do problema não são conhecidas, o que gera a necessidade de se realizar uma análise das causas com o objetivo de definir as ações corretivas mais adequadas. Uma forma conhecida de se desenvolver este tipo de atividade é a utilização do diagrama de espinha de peixe (ISHIKAWA, 1990). O objetivo deste diagrama é levantar todas as causas possíveis do problema para que, posteriormente, possam ser endereçadas adequadamente.

O problema é, portanto, algo que causa incômodo ao cliente e que causa nele o desejo de uma solução. No entanto, para resolver os problemas de um cliente não basta apenas conhecer seus problemas. É fundamental conhecer o contexto técnico ou de negócio em que ele se encontra para sugerir a solução mais adequada.

2.4.4 Domínio do Problema

O objetivo de uma pessoa ou empresa ao decidir adquirir um produto de *software* é ter seus problemas técnicos ou de negócios solucionados. Por esta razão, é parte do trabalho da equipe de desenvolvimento entender os problemas do cliente dentro de seu contexto para atender suas reais necessidades. É importante para a equipe de desenvolvimento ver claramente todas as questões pertinentes ao domínio do problema.

O domínio do problema é a área de conhecimento ou aplicação que necessita ser examinada para se resolver um problema. O domínio se refere exclusivamente aos tópicos relevantes para a solução do problema. Isso inclui os objetivos que o cliente deseja alcançar, o contexto no qual o problema existe e todas as regras que definem as funcionalidades de uma solução. O termo representa tanto o ambiente em que a solução será inserida quanto o problema em si.

Uma vez entendidos os problemas do cliente dentro do contexto do seu negócio é necessário partir para a análise das necessidades dos usuários do sistema. Estas necessidades, de maneira geral, são apresentadas pelo cliente como as soluções que ele imagina para seu problema no formato de um *software*.

2.4.5 Necessidades do Usuário

Do ponto de vista da Engenharia de Requisitos, pode-se dizer que as necessidades descrevem o que o cliente precisa de um sistema ou produto para resolver seus problemas. O termo Necessidade do Cliente (NC) é definido como “um reflexo de um problema de negócio, pessoal ou operacional (ou oportunidade) que justifica a compra ou uso de um novo sistema” (LEFFINGWELL & WIDRIG, 2003). Segundo Mcewen (2004), as necessidades descrevem o que o cliente necessita para resolver os problemas que encontra em sua vida pessoal ou no atendimento dos objetivos de sua organização.

Entender as necessidades dos usuários e outros interessados que serão afetados pela solução é uma responsabilidade dos analistas de negócios e analistas de sistemas envolvidos no projeto (LEFFINGWELL & WIDRIG, 2003). Para entender as necessidades do cliente é importante levar em conta que, normalmente, ele define as necessidades de acordo com sua interpretação de como resolver seus problemas por meio de um sistema. Assim, a definição das necessidades está limitada à capacidade do cliente em antever formas de resolução de seus problemas. Portanto, as necessidades definidas por ele podem ser inconsistentes com os problemas ou podem ser alternativas ruins de solução. Assim sendo, é importante analisar criteriosamente cada uma das necessidades definidas pelo cliente.

Documentar as necessidades do cliente envolve identificar, entender e representar diferentes pontos de vista. Cada envolvido percebe o problema de uma perspectiva diferente e é necessário entender as necessidades de cada um para entender o domínio do problema de uma maneira ampla. É fundamental para uma equipe de Engenharia de *Software* conhecer e conversar com os envolvidos no projeto para que possa conhecer seus pontos de vista. Além disso, necessidades dos envolvidos frequentemente podem ser conflitantes, como o custo máximo

definido por um *stakeholder* pode não ser suficiente para atender todo o escopo solicitado por outro.

Existem diversas formas de elicitar as necessidades dos envolvidos, como entrevistas, questionários e *storyboards*, *entre outros* (LEFFINGWELL & WIDRIG, 2003). A lista de necessidades obtidas a partir destas técnicas mostra o que o cliente deseja em alto nível e, em geral, de maneira ambígua. É necessário esclarecer e refinar esta lista de necessidades. Assim, nem todas as necessidades descritas em um primeiro momento serão traduzidas em funcionalidades do sistema. Algumas delas podem indicar restrições e atributos do sistema. Outras, por sua vez, podem ser eliminadas por causar contradições ou não fazer sentido.

Apesar do conceito de NC poder se confundir com o conceito de Necessidades Humanas e Organizacionais, eles são diferentes. Enquanto o primeiro trata das necessidades dos clientes no contexto de um sistema de *software* que solucione seus problemas, o segundo representa as necessidades básicas motivadoras, que precisam estar em equilíbrio para que não existam os problemas.

Finalmente, é importante ressaltar que, antes de passar para o domínio da solução, é crítico que o domínio do problema esteja completamente entendido (DORFMAN & THAYER, 1990). Essa compreensão do domínio do problema deve evitar que requisitos confusos e inadequados sejam definidos. A partir daí, as necessidades dos clientes podem ser traduzidas em Características do *software*, que são parte do domínio da solução.

2.4.6 Características do Sistema

O termo “característica” é pouco utilizado na ER e pode ser descrito como a forma como o cliente descreve o comportamento desejado de um sistema. Em geral o cliente traduz sua necessidade em um comportamento do sistema que ele espera que solucione seu problema. Na prática características não podem ser classificadas nem como necessidades nem como requisitos.

De uma forma mais técnica uma Característica pode ser definida como um serviço fornecido pelo sistema que preenche uma ou mais necessidades do cliente (LEFFINGWELL & WIDRIG, 2003). A diferença entre as necessidades e as

características é que necessidades não indicam uma solução particular, elas simplesmente descrevem algo que é necessário em um negócio (MCEWEN, 2004).

Dependendo do cliente, pode ser que ele expresse somente necessidades e nenhuma característica e vice-versa. Em outros casos pode ser que o analista obtenha um conjunto mesclado. Por esta razão é importante que estes profissionais estejam preparados para identificar e classificar as informações obtidas do usuário. Com o conjunto de necessidades do cliente e características de *software* devidamente organizado, pode-se iniciar a definição dos requisitos da solução.

2.4.7 Requisitos de Sistema

Requisitos são descrições mais técnicas e específicas que as características, estudadas na seção anterior. Um Requisito consiste na definição de uma propriedade ou comportamento que um produto ou sistema deve atender. O termo é amplamente difundido em Engenharia de *Software* e, de acordo com Dorfman e Thayer, é definido como:

“(1) Uma capacidade do *software* que o usuário necessita para resolver um problema ou alcançar um objetivo; (2) Uma capacidade de *software* que deve ser satisfeita para satisfazer um contrato, padrão, especificação ou outra documentação imposta formalmente.” (DORFMAN & THAYER, 1990)

Costuma-se classificar os requisitos em duas categorias: Requisitos Funcionais e Requisitos Não funcionais. Requisitos funcionais apresentam uma descrição completa de como o sistema funcionará, de acordo com a perspectiva do usuário. Requisitos não devem ser vistos como a descrição de uma máquina a ser construída, mas sim como uma descrição dos efeitos que o cliente espera que ela tenha sobre o mundo (JACKSON, Problems & Requirements, 1995).

Requisitos não funcionais, por outro lado, definem propriedades e impõem restrições ao projeto (MCEWEN, 2004). Em outras palavras, requisitos não funcionais especificam atributos do sistema, não o que ele irá fazer. Estes requisitos desempenham um papel crítico durante o desenvolvimento do sistema, servindo por vezes como critério de tomada de decisões. A omissão destes requisitos geralmente está entre os problemas mais caros e difíceis de corrigir uma vez que o sistema tenha sido concluído (MYLOPOULOS, CHUNG, & NIXON, 1992). Requisitos não

funcionais costumam ser classificados em algumas categorias como Usabilidade, Confiabilidade, Desempenho, Suportabilidade e Segurança.

As principais qualidades que se espera da documentação de especificação de requisitos são: objetividade, completude, consistência, rastreabilidade e ausência de detalhes de projeto (MCEWEN, 2004). Objetividade significa clareza de informação e ausência de ambiguidades na documentação. Completude significa conter todas as informações referentes ao requisito. Consistência significa ser isento de falhas e contradições. Rastreabilidade significa saber como os requisitos se relacionam entre si e quais elementos da arquitetura participam de sua *implementação*.

Outra questão importante na especificação de requisitos é a complexidade. Um requisito complexo deve ser dividido em requisitos mais simples e mais gerenciáveis (JACKSON, Problems & Requirements, 1995). Desta forma, obtém-se a decomposição de um problema complexo em problemas simples que se sabe como resolver.

Conforme discutido em seções anteriores, existem diversas técnicas para se identificar e descrever requisitos. Desta forma, os analistas de requisitos podem selecionar entre estas a que mais lhes convenha. Uma vez que os requisitos sejam identificados, é possível iniciar o desenvolvimento da arquitetura do produto. Entretanto, este tema não faz parte deste trabalho de pesquisa. A seguir são discutidos os principais problemas relacionados a projetos de sistemas de *software*.

2.5 Problemas Relacionados a Projetos de Software

A maior parte das outras disciplinas de engenharia possuem métodos muito bem estabelecidos, principalmente por serem disciplinas que vêm sendo desenvolvidas há muitos anos ou até mesmo séculos como, por exemplo, a Engenharia Civil. Por outro lado, a Engenharia de *Software* (ES) começou a ser desenvolvida há menos de meio século, com o surgimento dos primeiros problemas com o desenvolvimento de projetos de *software*.

As dificuldades dos desenvolvedores de *software* ficaram evidentes na literatura a partir do início da década de 70 pelo trabalho de (DIJKSTRA, 1972). Neste trabalho foram apontados como principais causas de problemas prazos e

orçamentos não condizentes com a realidade, sistemas de baixa qualidade, não atendimento dos requisitos, entre outros. Isso ajudou a deixar claro que, ao contrário do que se pensava, projetar *software* é uma atividade bastante complexa.

Muito desta complexidade se deve à própria natureza do *software*. Em geral, esses sistemas são mais complexos em relação a seu tamanho, que qualquer outro projeto de engenharia (BROOKS Jr., 1987). Outra questão que torna complexo o desenvolvimento de *software* é a diversidade da natureza dos problemas que são resolvidos por meio destes sistemas. Enquanto a Engenharia Mecânica concentra-se em projeto de sistemas mecânicos como máquinas e veículos, a ES pode desenvolver aplicações para todas as áreas do conhecimento humano, como Direito, Administração, Medicina, Aeronáutica, só para citar alguns exemplos.

O fato da ES ter um campo de atuação tão amplo parece ser uma das causas da falta de maturidade desta disciplina. Enquanto para as outras engenharias existem uma delimitação mais clara dos problemas a serem solucionados, na ES o número de problemas a serem resolvidos parece ser ilimitado o que dificulta a formação de conhecimentos sólidos.

A infinidade de possibilidades aliada a pouca idade da ES faz com que, nesta área, as soluções ainda não sejam bem estabelecidas, assim como os processos e técnicas utilizados. Têm sido desenvolvidas técnicas, processos, modelos de maturidade ao longo dos últimos 40 anos. Apesar de tudo isso, os projetos de *software* costumam ainda ser deficitários em relação à qualidade e satisfação das expectativas do cliente. Muitos projetos passam ainda por problemas no cumprimento dos prazos e, em consequência disso, extrapolam os custos estimados.

De acordo com as estatísticas apresentadas pelo *Chaos Report* (DOMINGUEZ, 2009), os números de projetos de sistemas de *software* mal sucedidos são alarmantes. O relatório de 2009 demonstra que os projetos de *software* têm percentuais de sucesso que giram em torno de 30%. Ainda, de acordo com dados do *Chaos Report*, em média apenas 52% das características e funcionalidades de um *software* são entregues, no caso dos projetos entregues com alteração de custos ou prazo.

Em estudos relacionados ao desenvolvimento de sistemas de *software* foram identificadas fortes evidências de que os problemas mais graves de projetos estão diretamente relacionados à especificação de requisitos (LEFFINGWELL &

WIDRIG, 2003). Um destes estudos, desenvolvido pela empresa de consultoria AIG, relaciona os resultados obtidos e projetos com a qualidade da especificação de requisitos (ELLIS, 2008). A diferença apontada é gritante, apenas para citar um exemplo, o percentual de projetos considerados bem sucedidos é de 43% nas empresas que desenvolvem excelentes técnicas de levantamento de requisitos enquanto na média das empresas estes valores giram em torno de 9%. Isso aponta para o problema de falta de maturidade de requisitos na grande maioria das empresas (ELLIS, 2009).

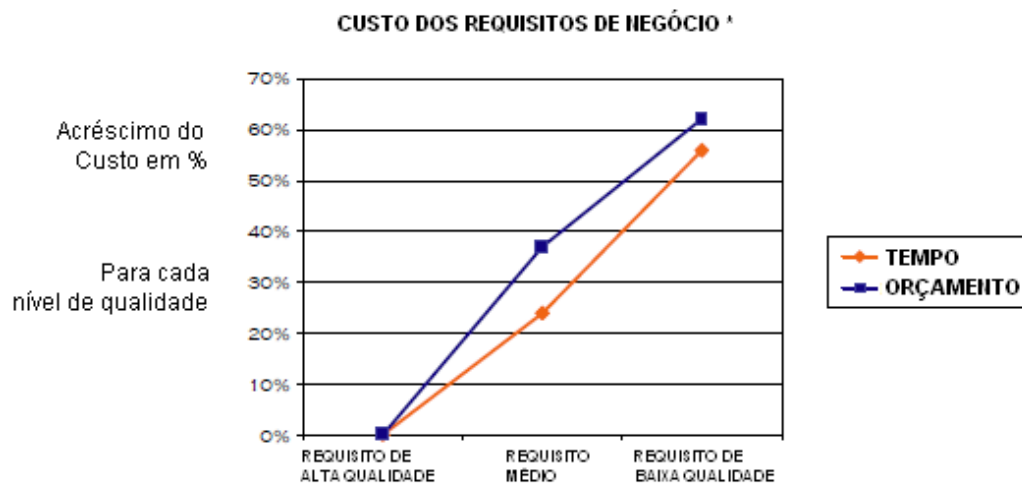
Tabela 2 - Distribuição dos Fatores de Falhas em Projetos

Ordem	Descrição	Percentual
1	Especificação de requisitos incompleta	13,1%
2	Falta de informação fornecida pelo usuário.	12,4%
3	Falta de recursos	10,6%
4	Expectativas não realísticas	9,9%
5	Falta de suporte executivo	9,3%
6	Mudança de requisitos	8,7%
7	Falta de planejamento	5,9%
8	Não precisa mais daquilo	5,3%
9	Falta de gestão da TI	4,3%
10	Analfabetismo tecnológico	3,7%
11	Outros	9,9%

Fonte: adaptado de (THE STANDISH GROUP, 2009)

Os problemas relacionados à ER somam 34,2% dos principais fatores falha em projetos, segundo dados do Chaos Report 2009 (THE STANDISH GROUP, 2009), porém não são os únicos. Outros fatores citados são relacionados a questões como falta de recursos, de planejamento e gestão de TI, entre outros conforme detalhado na Tabela 2.

Os problemas discutidos geram dificuldades de ordem econômica devido aos prejuízos que causam (LEFFINGWELL & WIDRIG, 2003). Os resultados da pesquisa *Business Analysis Benchmark* de 2008 demonstram que as empresas que não usam boas práticas de especificação requisitos pagam um preço caro por isso (ELLIS, 2008). Os projetos com requisitos de baixa qualidade chegam a gastar sessenta por cento mais tempo e dinheiro que projetos com requisitos de alta qualidade, como ilustra a Figura 14.



* Acréscimo médio no valor e no tempo gastos em projetos de acordo com a qualidade dos requisitos.

Figura 14 - Custo da Má Especificação de Requisitos
 Fonte: Adaptado de (ELLIS, 2008).

Ainda em relação ao preço pago pela má especificação de requisitos, estudos realizados em diversas companhias mostram que o custo de correção de problemas aumenta com o passar do tempo conforme ilustra a Figura 15. Por exemplo, o custo de reparar uma falha na fase de codificação é cerca de cinco a dez vezes maior que na fase de requisitos (DAVIS, 1993). Além disso, este custo é cerca de 20 vezes menor que na fase de manutenção.

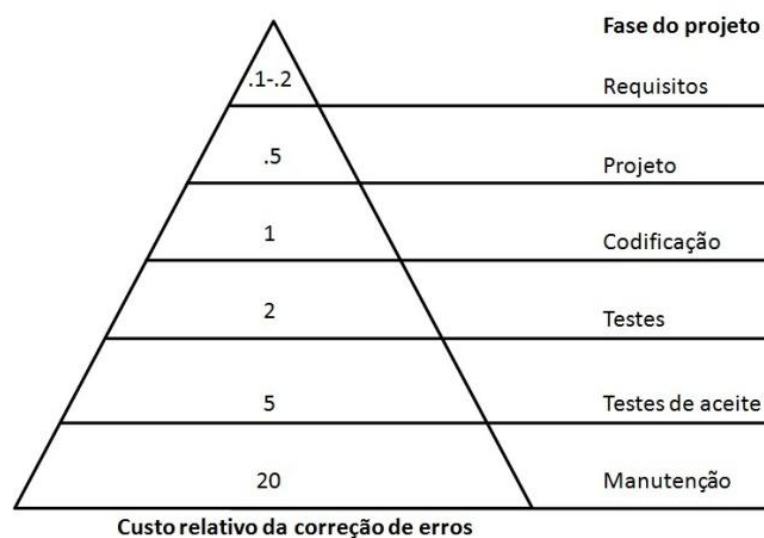


Figura 15 - Custo de Correção de Defeitos em Diferentes Fases do Projeto
 Fonte: (DAVIS, 1993)

As informações apresentadas nesta Seção demonstram o quanto a ER e a ES ainda necessitam de avanços em seus métodos e técnicas. A resposta para esse problema pode estar tanto na definição de métodos mais eficientes quanto na maturidade das empresas no uso de processos.

2.6 Conclusões

Este Capítulo apresentou uma discussão sobre o que é Engenharia de Sistemas e quais as ferramentas que podem ser utilizadas em sua modelagem. Foram apresentados os principais conceitos de Engenharia de *Software*, suas características, principais processos e métodos. Além disso, foram discutidos os processos de especificação de requisitos e suas etapas. Foram apresentadas algumas técnicas de especificação de requisitos, como quadros de problema, e discutidas suas principais vantagens e desvantagens.

Dentre as técnicas de especificação de requisitos apresentadas neste capítulo nenhuma consegue alcançar tanto a análise dos problemas quanto o estudo das necessidades do cliente. Uma técnica que abranja estas duas fases pode trazer resultados positivos para a especificação de requisitos, no sentido de reduzir os problemas decorrentes de erros e falhas. Como visto na seção final, os problemas decorrentes da má especificação de requisitos estão entre os mais graves dos projetos de *software*. Portanto, uma técnica que ofereça melhorias neste processo é uma contribuição importante pra a ES.

A Teoria de Projeto Axiomático (TPA) é uma técnica para a elaboração de projetos que tem por objetivo a tomada das decisões corretas no planejamento de um sistema. Ela pode ser aplicada em praticamente qualquer área de conhecimento, inclusive na especificação de projetos de *software*. Até o momento existem alguns trabalhos relacionados ao desenvolvimento de sistemas de *software* associado utilizando conceitos de TPA. No entanto, não se tem conhecimento de nenhum trabalho que aplique TPA à especificação de requisitos.

3 Projeto Axiomático

Neste Capítulo é introduzida a Teoria de Projeto Axiomático (TPA), contexto histórico da sua criação e seus principais objetivos e características. Além disso, apresenta-se um resumo geral dos principais conceitos da teoria. Neste Capítulo também são discutidos os trabalhos relacionados com TPA e desenvolvimento de *software*. O capítulo culmina na apresentação de uma Abordagem de Projeto de Software Orientado a Objetos baseado na TPA, que inspirou o desenvolvimento deste trabalho de mestrado.

3.1 Introdução à Teoria de Projeto Axiomático

O conceito de Projeto Axiomático surgiu para a comunidade no início dos anos 90 (SUH, 1990) a partir da percepção da necessidade de se estabelecer uma base científica para projetos na área de manufatura que, até então, costumavam ser baseados no conhecimento empírico de quem os elaborava. Nam Suh, pesquisador do departamento de engenharia mecânica do *Massachusetts Institute of Technology* (MIT), buscou estabelecer axiomas que fossem comuns a qualquer projeto.

Do ponto de vista da lógica matemática, os axiomas são verdades que não podem ser derivadas mas para as quais não existem contraexemplos e exceções (SUH, 1990). Para outras áreas do conhecimento como física, química e biologia, os axiomas representam princípios fundamentais, como por exemplo as leis da mecânica de Newton.

Suh observou em seus estudos que a maioria dos projetos da área de mecânica, assim como os de diversas outras áreas, eram elaborados com base no empirismo. Por esta razão, decidiu aprofundar o estudo deste assunto. Inicialmente, trabalhou com a seguinte pergunta: “Dado um conjunto de requisitos funcionais para um determinado produto, existem axiomas de aplicação genérica que levam a decisões corretas para planejar um sistema de produção ótimo?”. Foram elaborados, então, 12 axiomas para responder a esta pergunta. Estes axiomas foram testados e avaliados, sendo finalmente reduzidos a apenas dois, mais um conjunto de corolários e teoremas (ver Anexo A).

O primeiro axioma é chamado Axioma da Independência. Este axioma diz que a independência dos requisitos funcionais deve sempre ser mantida. O segundo axioma é chamado de Axioma da Informação e diz que, entre aqueles projetos que satisfaçam o primeiro axioma, o melhor será aquele que possuir o menor conteúdo de informação.

O objetivo da TPA é estabelecer uma base científica para projeto de forma que as atividades sejam baseadas em uma fundamentação teórica (SUH, 2001) e não mais no empirismo. Além disso, Suh(2001) afirma que o projeto axiomático aprimora a criatividade do projetista porque exige uma definição clara dos objetivos do projeto e fornece critérios para que as ideias ruins sejam eliminadas o mais cedo possível. Um dos conceitos fundamentais formulados por Suh para estabelecer esta base científica para projetos é o de domínios de projeto. Segundo ele existem quatro domínios fundamentais que contribuem no desenvolvimento de um projeto conforme descrito na seção a seguir.

3.2 Domínios de Projeto

De acordo com Suh (2001), o mundo de projeto consiste de quatro domínios: Domínio do Cliente, Domínio Funcional, Domínio Físico e Domínio de Processo. Estes domínios delimitam quatro tipos diferentes de atividades de projeto que se inter-relacionam. Neste relacionamento, um domínio representa “o que queremos alcançar” e o outro “como iremos alcançar”. Na Figura 16 pode ser vista uma representação destes domínios e seus relacionamentos.

O Domínio do Cliente é caracterizado pelas Necessidades do Cliente (NCs) e no Domínio Funcional essas necessidades são traduzidas em Requisitos Funcionais (RFs) do produto. Para satisfazer os requisitos funcionais são concebidos os Parâmetros de Projeto (PPs) no Domínio Físico. Para produzir o produto especificado pelos PPs é desenvolvido um processo caracterizado pelas Variáveis de Processo, pertencentes ao Domínio de Processo (SUH, 2001).

Os conceitos de projeto axiomático são, principalmente, aplicados no relacionamento entre os domínios funcional e físico, pois a atividade de projeto em si é representada pelo mapeamento de requisitos funcionais para parâmetros de

projetos. Neste relacionamento, o domínio funcional representa “o que queremos alcançar” e o físico “como iremos alcançar”.

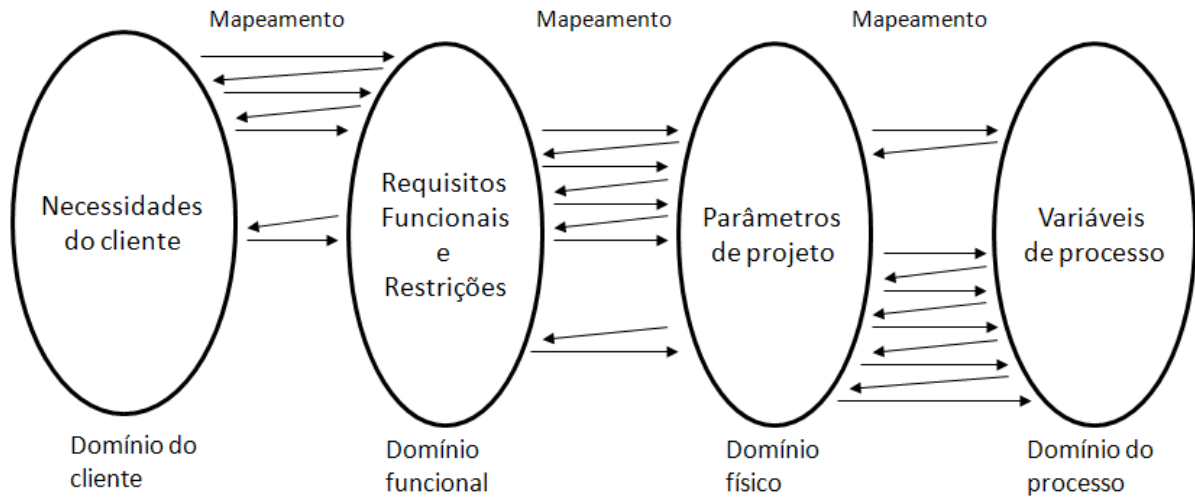


Figura 16 - Domínios de Projeto
Fonte: (SUH, 2001)

Suh propõe um processo de projeto no qual os requisitos funcionais do produto são relacionados com parâmetros de projeto, esse processo é chamado de mapeamento. Após a realização do mapeamento entre domínios funcional e físico, a definição do projeto segue para uma decomposição dos requisitos funcionais e, novamente, seu mapeamento para parâmetros de projeto devidamente decompostos.

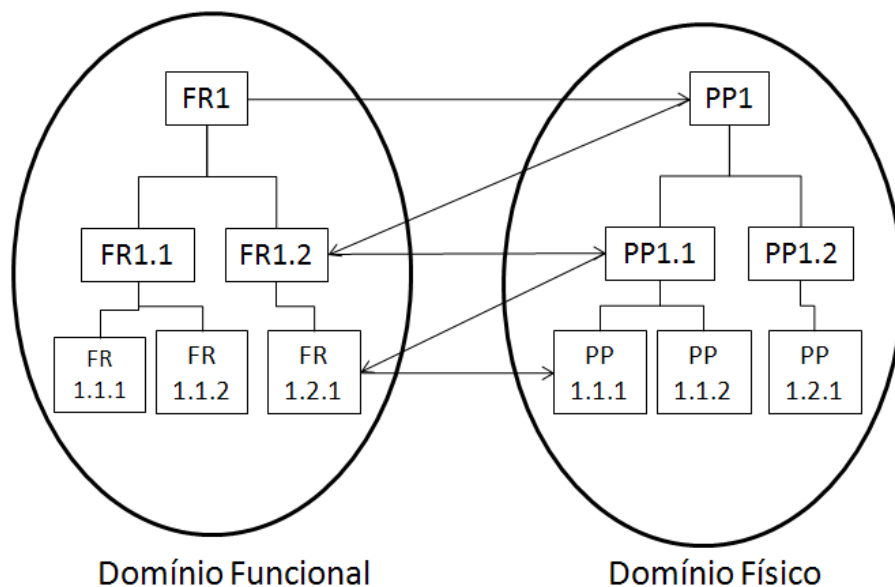


Figura 17 - Processo em Zig-zag
Fonte: (SUH, 2001)

Este processo, denominado Processo em *Zig-zag*, pode se repetir até que não seja mais possível decompor os itens pertencentes ao domínio funcional. A Figura 17 ilustra o processo de decomposição em *zig-zag* entre o domínio funcional e o domínio físico.

O processo de mapeamento tem como principal objetivo verificar o nível de dependência entre os parâmetros de projeto em relação aos requisitos funcionais. Essa necessidade advém do primeiro axioma da TPA, denominado axioma da independência.

3.3 Axioma da Independência

O Axioma da Independência (AI) define que quando existem dois ou mais requisitos funcionais, cada um desses requisitos funcionais deve ser satisfeito sem afetar os outros (SUH, 2001). Isto significa que os parâmetros de projeto devem ser identificados de forma a satisfazer os requisitos funcionais e manter sua independência.

A forma utilizada para se verificar a adequação do projeto ao primeiro axioma é a utilização de matrizes de projeto em que os requisitos são listados como linhas e os parâmetros de projeto como colunas. Para demonstrar que existe um relacionamento entre um requisito e um parâmetro de projeto, marca-se um “X” na célula em que se cruzam. O “X” nas células significa que o PP é usado totalmente ou parcialmente para realizar o RF, sendo que “X” indica um valor diferente de zero.

Em um projeto ideal a matriz de projeto é diagonal, ou seja, cada RF se relaciona somente com um PP conforme ilustrado na Figura 18. Um projeto que apresente este tipo de matriz é denominado “Desacoplado”.

	PP1	PP2	PP3
RF1	X	0	0
RF2	0	X	0
RF3	0	0	X

Figura 18 - Matriz de Projeto Desacoplada
Fonte: (SUH, 1990)

Os elementos da matriz localizados acima da linha diagonal são chamados triângulo superior e os elementos localizados abaixo da mesma linha são chamados triângulo inferior. Se um destes triângulos possuir elementos marcados com “X”, esta matriz é dita triangular superior ou inferior. Sobre a matriz de projetos também é importante dizer que, segundo o Teorema 4 (ver Anexo A), o número de RFs deve ser igual ao número de PPs, ou seja, a matriz deve ser quadrada.

Embora a matriz de projeto ideal seja diagonal, a forma mais comum de ocorrer é a triangular semelhante à apresentada na Figura 19. Uma matriz triangular se caracteriza por ter elementos marcados com “X” somente no seu lado inferior ou somente no seu lado superior. Um projeto que possui uma matriz com esta característica é dito “Semiacoplado”.

A existência destes relacionamentos adicionais indica que um RF é satisfeito por mais de um PP. Mesmo não sendo o tipo de projeto ideal, este tipo de projeto é aceitável por permitir ao projetista manter o controle sobre o projeto. No entanto, por definição, um projeto semiacoplado satisfaz o AI somente se a estrutura da matriz é conhecida e a sequência ditada por ela para realização dos PPs é seguida (LEE & JEZIOREK, 2006).

	PP1	PP2	PP3
RF1	X	0	0
RF2	X	X	0
RF3	X	X	X

Figura 19 - Matriz de Projeto Semiacoplada
Fonte: (SUH, 1990)

Outra informação importante para se identificar uma matriz semiacoplada é que uma matriz que apresente elementos tanto na parte superior quanto na inferior pode ser reorganizada. Caso seja possível reordenar os elementos fazendo com que todos os elementos fiquem do mesmo lado, a matriz será considerada semiacoplada, caso contrário será considerada acoplada.

Uma matriz de projeto acoplada é inadequada e deve sempre ser eliminada ou, ao menos, redefinida caso identificada. Este tipo de matriz de projeto é representada na Figura 20. Neste caso, os relacionamentos entre RFs e PPs não se

limitam apenas à porção inferior da matriz de projeto. Esta situação é problemática por que caso ocorra uma mudança, por exemplo, em RF1 esta implicará na alteração em PP1. No entanto como PP1 também participa na realização de outros RFs essa alteração no PP1 poderá desencadear modificações em várias outras partes do projeto. Desta forma, as alterações realizadas em um PP ou RF tendem a refletir em diversas partes do projeto de uma forma cíclica, causando problemas complexos e difíceis de resolver.

	PP1	PP2	PP3
RF1	X	X	X
RF2	X	X	X
RF3	X	X	X

Figura 20 - Matriz de Projeto Acoplada
Fonte: (SUH, 1990)

Como a TPA pretende ser uma ferramenta para a tomada de decisões de projeto, pode-se elaborar diversas soluções de projeto e escolher aquela que melhor se adéqua ao AI. Para escolher entre estas diversas soluções podem ser utilizadas duas medidas de independência funcional: a reangularidade e a semangularidade (SUH, 2001) (OLEWNIK & LEWIS, 2003).

Nos trabalhos de Suh (2001) e Olewnik (OLEWNIK & LEWIS, 2003) são definidos os conceitos e apresentadas fórmulas para o cálculo da reangularidade e a semangularidade (ver Apêndice A). Ambas, possuem um valor máximo de 1, que corresponde a um projeto desacoplado (ideal) e, também, ambas possuem um valor 0 para projetos acoplados (inaceitáveis). Assim, a independência funcional é proporcional ao valor de reangularidade e semangularidade calculados para a matriz de projeto.

Desta forma, ao se comparar duas ou mais soluções de projeto, aquela que tem os maiores fatores de reangularidade e semangularidade possui maior independência funcional e, portanto, é a escolha. Caso estes valores sejam muito semelhantes pode-se usar, como fator de escolha entre dois projetos, o Axioma da Informação.

3.4 Axioma da Informação

O segundo axioma, ou Axioma da Informação (AInf), estabelece que “o melhor projeto é um projeto funcionalmente desacoplado que tem o conteúdo de informação mínimo”(SUH, 2001). Em um enunciado mais simples, o AInf consiste em “Minimizar o conteúdo de informação do projeto” (SUH, 2001). O conteúdo de informação pode auxiliar na identificação do conjunto de parâmetros de projeto mais adequado para satisfazer um determinado conjunto de requisitos funcionais.

Informação é definida como a medida do conhecimento necessário para satisfazer um RF em um determinado nível da hierarquia de RFs (SUH, 2001). Desta forma, quanto maior a informação necessária para satisfazer os requisitos funcionais, mais difícil será para realizar o projeto com sucesso, pois a complexidade do projeto e do seu desenvolvimento aumentará (PIMENTEL, 2007). Assim, o conteúdo de informação está diretamente relacionado com a probabilidade de sucesso no projeto e desenvolvimento de um produto.

3.5 Projeto Axiomático e Desenvolvimento de *Software*

Desde sua criação, o projeto axiomático vem sendo aplicado nas mais diversas áreas, como projeto de produtos, processos de manufatura e organização de empresas. Isso se deve ao fato do projeto axiomático ser independente de domínio, o que o torna aplicável também no desenvolvimento de *software*. O primeiro trabalho no qual foi utilizado o Projeto Axiomático no desenvolvimento de *software* foi apresentado em 1991 (KIM; SUH; KIM, 1991).

Por outro lado, no desenvolvimento de *software* orientado a objetos usando o Projeto Axiomático, um dos primeiros trabalhos foi apresentado por Do e Suh (1999). Este trabalho propôs encontrar os componentes que serão *implementados* a partir das necessidades do cliente, definindo, assim, as classes e objetos do sistema. O processo de desenvolvimento axiomático de *software* é representado por um diagrama em V, como o ilustrado na Figura 21.

Outro trabalho, também relacionado ao desenvolvimento de *software*, demonstra o uso da teoria de projeto axiomático para o gerenciamento do processo de desenvolvimento com o objetivo de garantir a qualidade do projeto e do produto

(DO, 2004). É apresentada uma extensão da teoria de projeto axiomático, que introduz o conceito de domínios complementares, que são adicionados ao modelo para endereçar necessidades específicas da área a que pertence o domínio do problema. No caso do desenvolvimento de *software*, o autor considera que casos de uso são uma forma simples de se definir e gerenciar os requisitos de um sistema. Por este motivo, recomenda que seja estabelecido um domínio de casos de uso com objetivo de modelar a natureza dinâmica dos requisitos de *software*, é dizer, modelar o comportamento do sistema com objetivo de atender cada requisito. Neste modelo os requisitos do cliente (*customer needs*) dirigem os casos de uso que, por sua vez, dirigem os requisitos funcionais e outros artefatos de decomposição.

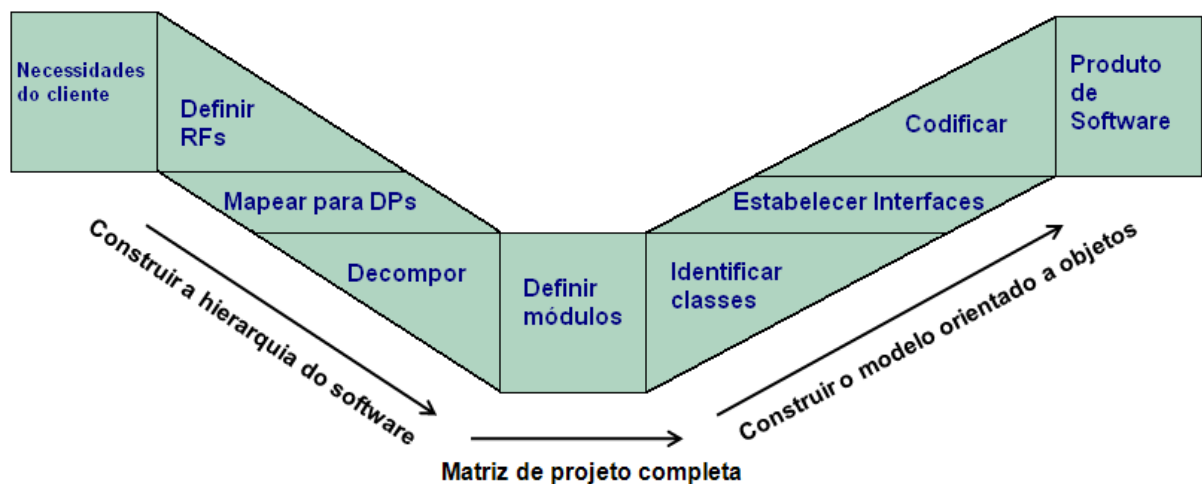


Figura 21 - Processo de Desenvolvimento Axiomático de *Software*
 Fonte: adaptado de (DO & SUH, 1999)

Mais recentemente, em (PIMENTEL, 2007), foi apresentada uma nova abordagem de projeto de *software* orientado a objetos baseado na teoria de projeto axiomático. Neste trabalho são utilizados conceitos fundamentais da UML como casos de uso, colaborações, classes, objetos, diagramas de atividade e de sequência. Casos de uso são utilizados para representar requisitos funcionais e colaborações para representar os parâmetros de projeto durante o processo de decomposição funcional. Neste trabalho são sugeridas diversas métricas para o cálculo do conteúdo da informação, estas métricas, inclusive, variam de acordo com o nível de decomposição. Esta abordagem será considerada com mais detalhes na próxima seção.

3.6 Desenvolvimento Orientado a Objetos Baseado em Projeto Axiomático

Nesta Seção é descrita a abordagem de projeto de *software* proposta por (PIMENTEL, 2007), que aplica a Teoria de Projeto Axiomático (TPA) ao desenvolvimento de *software* orientado a objetos baseado no Processo Unificado (PU). Inicialmente é apresentada uma introdução à abordagem proposta, incluindo os principais conceitos em que se baseia. Além disso, estabelecem-se as principais correspondências entre conceitos de orientação a objetos e projeto axiomático, de acordo com esta abordagem. Na sequência, é estabelecido o relacionamento entre os domínios de projeto e as fases do projeto unificado. Também é discutida a aplicação do primeiro axioma a projetos de *software* de acordo com a abordagem. Finalmente, são apresentadas as métricas propostas para aplicação do segundo axioma a projetos de *software*.

3.6.1 Aspectos Gerais

A abordagem de Pimentel (2007), baseada em TPA E POO, tem por objetivo trazer vantagens ao processo de desenvolvimento de *software*. Entre estas vantagens está o estabelecimento de critérios para a tomada de decisões de projeto como escolher o conjunto mais apropriado de requisitos funcionais e escolher a melhor solução de projeto entre as possíveis maneiras de se modelar a solução.

A TPA é comprovadamente eficaz na melhoria de resultados na elaboração de projetos em diversas áreas e pode ser utilizada, inclusive, no desenvolvimento de *software*. Por sua vez, o PU é um dos processos mais difundidos de desenvolvimento de *software*. PU é utilizado, na maioria das vezes, em conjunto com a UML, a linguagem de modelagem orientada a objetos mais utilizada atualmente. Por esta razão, ao se pensar em uma abordagem de projeto axiomático de *software* orientado ao objetos, parece um caminho natural integrar estas técnicas. Para isso é necessário estabelecer relações entre as fases do PU e os mapeamentos entre domínios da TPA e, também, estabelecer correspondências entre os conceitos básicos de TPA e UML. Por esta razão, são apresentadas a seguir as definições propostas por (PIMENTEL, 2007) no contexto da UML:

Caso de Uso: um caso de uso é a especificação de um conjunto de ações que representa uma utilização completa do sistema por um ou mais atores.

Subcaso de Uso: um subcaso de uso é a especificação de um conjunto de ações que representa uma utilização não completa do sistema ou subsistema. Um subcaso de uso representa uma parte de uma utilização completa do sistema, ou seja, parte de um caso de uso.

Subcaso de uso independente: um subcaso de uso independente de características técnicas é a especificação de um conjunto de ações que um sistema executa que representa uma utilização não completa do sistema que é especificada sem levar em consideração características técnicas empregadas na solução.

Subcaso de uso dependente: um subcaso de uso dependente de características técnicas é a especificação de um conjunto de ações que um sistema executa que representa uma utilização não completa do sistema que é especificada considerando fortemente características técnicas empregadas na solução.

Serviço Técnico: um serviço técnico é um serviço esperado por um objeto, classe ou componente do sistema de outro objeto, classe ou componente do sistema.

Colaboração: uma colaboração é o conceito da UML responsável por representar a realização de um caso de uso (BOOCH, RUMBAUGH, & JACOBSON, 2006).

Uma vez definidos os conceitos da UML utilizados na abordagem, é possível estabelecer sua relação com os conceitos de requisitos funcionais (RFs) e parâmetros de projeto (PPs) definidos por Suh (SUH, 2001).

De acordo com Suh, os RFs representam as características funcionais esperadas de um produto (SUH, 2001). Por sua vez, Booch, Rumbaugh e Jacobson (2006) definem um caso de uso como a representação completa de um requisito funcional do sistema. Assim sendo, casos de uso podem representar o conceito de requisito funcional do Projeto Axiomático como ocorre na abordagem de Pimentel (2007).

Ainda, segundo Booch, Jacobson e Rumbaugh (2006), as colaborações são utilizadas para especificar a realização de casos de uso e operações. A relação de realização entre casos de uso e colaborações é ilustrada na Figura 22. Por outro

lado, na TPA os PPs são especificados para satisfazer os RFs do projeto. Portando, isto significa que, em um contexto em que Casos de Uso representam requisitos funcionais, as colaborações podem ser utilizadas para representar os PPs.

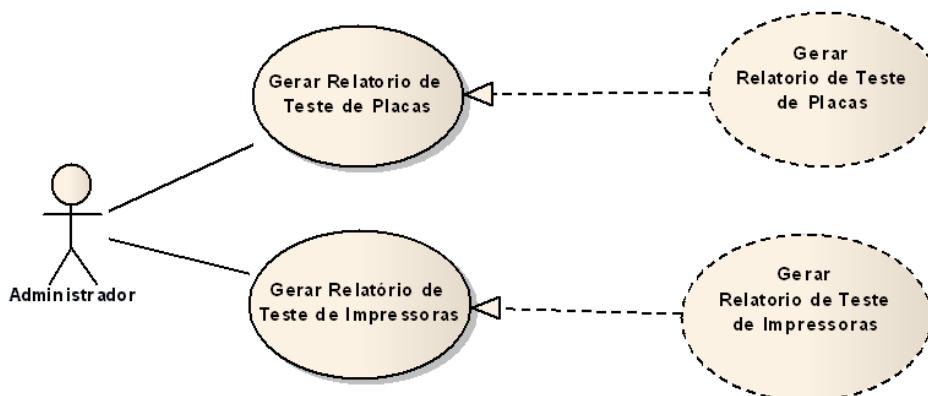


Figura 22 - Realização de Casos de Uso
Fonte: Autoria própria

Esta representação é válida em níveis de abstração mais altos do projeto mas, conforme os requisitos funcionais são decompostos, é necessário utilizar outros elementos. Desta maneira, casos de uso são decompostos em subcasos e as colaborações em subcolaborações. As subcolaborações representam interações menores que compõem a colaboração original. A Figura 23 apresenta um exemplo de decomposição de Casos de Uso e Colaborações. Neste exemplo o caso de uso “Testar Placa” está composto em dois subcasos, “Testar Sensor de Papel” e “Testar Memória”, que são realizados por subcolaborações identificadas com os mesmos nomes respectivamente.

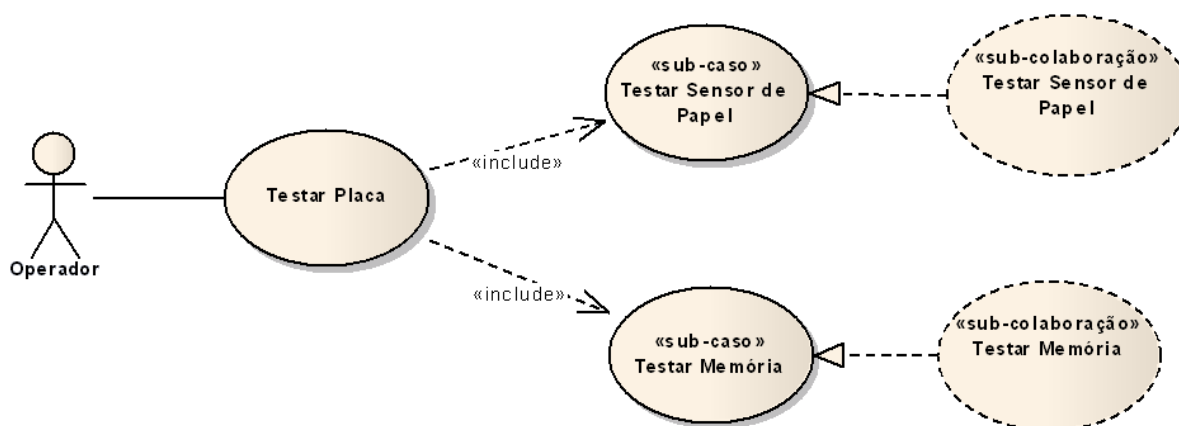


Figura 23 - Decomposição de Casos de Uso e Colaborações
Fonte: Autoria própria

Esta divisão ou detalhamento dos casos de uso e colaborações em elementos menores é necessária para que seja possível adaptar a abordagem proposta ao processo em *zig-zag* (SUH, 2001). De acordo com este processo, os RFs devem ser refinados ou decompostos e, a cada iteração, devem ser identificados os PPs correspondentes. Com vistas a realizar este processo de decomposição e mapeamento entre os domínios funcional e físico, foram definidos níveis de hierarquia funcional para os casos de uso, chegando-se à conclusão de que quatro níveis com uma ou mais decomposições cada um são suficientes para o detalhamento de um projeto de *software* (PIMENTEL, 2007). A Tabela 3 demonstra as correspondências de RFs e PPs com conceitos da UML em cada nível de decomposição.

Tabela 3 - Níveis de Abstração

Nível de Abstração	Requisitos Funcionais (RF)	Parâmetros de Projeto (PP)
1	Casos de Uso	Colaborações e seus papéis
2	Subcasos de uso independentes de características técnicas.	Subcolaborações e seus papéis
3	Subcasos de uso dependentes de características técnicas.	Subcolaborações e seus papéis
4	Serviços técnicos	Objetos e classes

Fonte: (PIMENTEL, 2007)

Uma vez identificadas as correspondências dos conceitos de requisito funcional e parâmetro de projeto com os elementos da UML, foi estabelecida uma relação de correspondência entre os processos de mapeamento entre domínios da TPA e as fases do PU.

3.6.2 Domínios de Projeto e Fases do Processo Unificado

O padrão do *Institute of Electrical and Electronics Engineers* (IEEE) para a criação de modelos de ciclo de vida de *software* estabelece que os principais processos da fase de desenvolvimento de *software* são: Requisitos, Projeto e *Implementação* (IEEE, 1997). No processo unificado estes processos estão distribuídos nas fases de concepção, elaboração e construção, além de ser definida uma quarta fase, a transição.

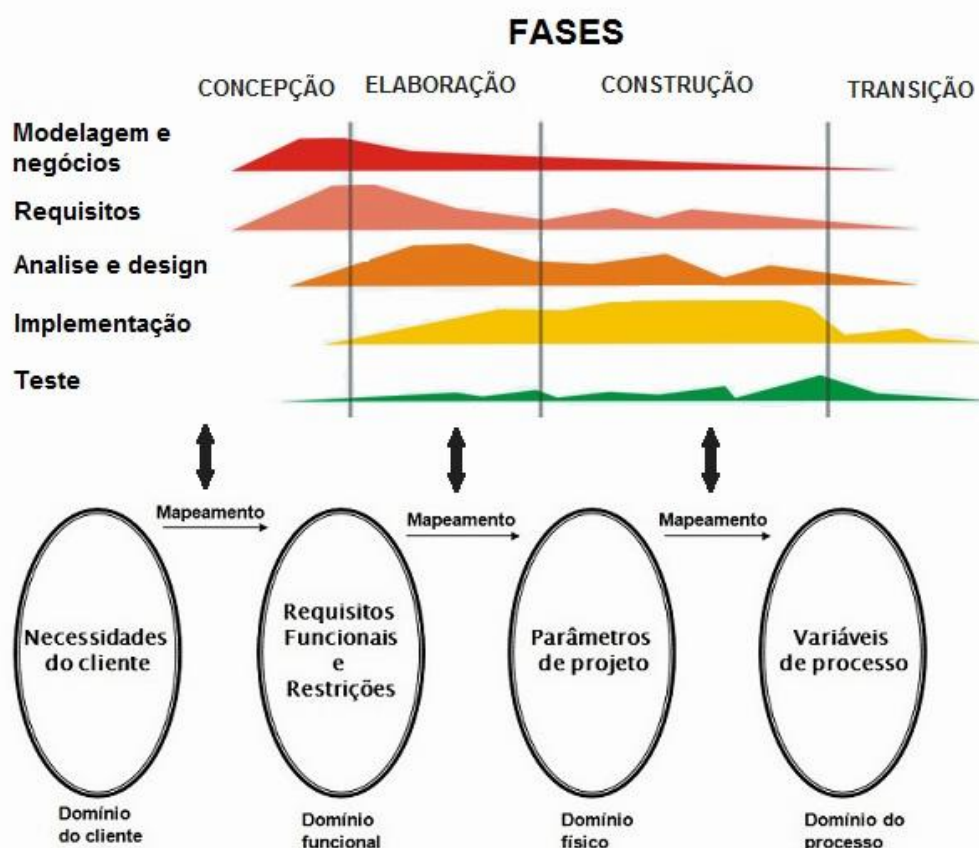


Figura 24 - Domínios de Projeto Axiomático e as Fases do Processo Unificado
 Fonte: Adaptado de (PIMENTEL, 2007)

Estudando o conceito de domínios de projeto definido por Suh (2001) e o processo de mapeamento entre domínios, nota-se que é possível relacionar estes mapeamentos e as fases definidas no PU conforme ilustra a Figura 24. No contexto de projeto de *software*, o mapeamento entre domínio do cliente e domínio funcional corresponde à fase de concepção do processo unificado, em que as necessidades do cliente são identificadas e convertidas em requisitos funcionais. Por sua vez, o mapeamento entre o domínio funcional e o domínio físico corresponde à fase de elaboração do processo unificado. Nesta fase os requisitos funcionais são transformados em parâmetros de projeto que, no caso de *software*, correspondem à arquitetura do sistema. Por fim, o mapeamento entre domínio físico e domínio de processo corresponde à fase de construção do processo unificado. Fase em que os requisitos funcionais são convertidos em variáveis de processo, o que corresponde aos códigos fonte do sistema.

Assim, com as relações estabelecidas entre elementos da TPA e alguns dos conceitos mais conhecidos do desenvolvimento de *software*, Pimentel (2007) passa

à discussão a respeito da aplicação do Axioma da Independência em projetos de *software*.

3.6.3 Aplicação do Axioma da Independência

Na abordagem de programação orientada a objetos baseada em projeto axiomático, a atividade de aplicação do Axioma da Independência (AI) deve ser realizada pelo menos uma vez a cada uma das etapas de decomposição apresentadas na Seção 3.6.1. Esta aplicação visa determinar se a solução elaborada para o projeto é adequada ou, caso tenham sido elaboradas diversas soluções de projeto, determinar qual delas deve ser adotada. Para escolher entre estas diversas soluções pode-se utilizar a reangularidade e a semangularidade como medidas de independência funcional (PIMENTEL, 2007). As equações utilizadas para estas métricas são apresentadas no Apêndice A.

Caso tenha sido elaborada uma solução de projeto adequada, de acordo com o primeiro axioma, pode-se seguir para a próxima etapa de decomposição hierárquica. Caso contrário, as soluções de projeto acopladas devem ser descartadas pois podem causar efeitos indesejáveis no desenvolvimento do projeto. Desta forma, é necessário elaborar uma nova solução ou, ao menos, reformular a solução acoplada de forma que atenda o AI.

Na abordagem, a matriz de projeto é utilizada para mapear os relacionamentos entre os RFs e os PPs e para verificar se o projeto é aceitável, ou seja, se é desacoplado ou semiacoplado. A Figura 25 apresenta um exemplo de uma matriz de projeto de um sistema de teste de impressoras. Como é possível observar, cada um RFs se relaciona somente com um PP, formando uma matriz diagonal. Assim, o RF2 (Caso de uso Testar Guilhotina) depende apenas de PP2 (Colaboração Testar Guilhotina) para ser atendido, o mesmo ocorre com todos os outros requisitos. No caso desta matriz, o valor da reangularidade é igual a 1, o que a torna uma solução de projeto ótima.

Tendo sido estudada a aplicação do AI no desenvolvimento de *software* resta apenas estudar uma forma para a aplicação do segundo axioma neste tipo de projetos. A solução encontrada foi utilizar métricas utilizadas, até então, nos projetos de *software*.

	PP							
RF								
	CO::Gerar Relatório de Teste							
	CO::Testar Guilhotina							
	CO::Testar Impressão de Caracteres							
	CO::Testar Impressão de Imagens							
	CO::Testar Interface Serial							
	CO::Testar Memória							
	CO::Testar Sensor de Papel							
	CO::Testar Sensor de Tampa							
UC::Gerar Relatório de Teste	X							
UC::Testar Guilhotina		X						
UC::Testar Impressão de Caracteres			X					
UC::Testar Impressão de Imagens				X				
UC::Testar Interface Serial					X			
UC::Testar Memória						X		
UC::Testar Sensor de Papel							X	
UC::Testar Sensor de Tampa								X

Figura 25 - Exemplo de Matriz de Projeto Desacoplada
Fonte: Autoria Própria

3.6.4 Aplicação do Axioma da Informação

De maneira geral, nos projetos desenvolvidos utilizando a TPA o conteúdo de informação pode ser medido a cada decomposição dos requisitos funcionais. Por esta razão, foram definidas métricas a serem utilizadas em cada uma das etapas do método proposto (PIMENTEL, 2007). Na primeira e segunda etapa o conteúdo é calculado utilizando-se a métrica de pontos por caso de uso (ANDA, DREIEM, SJOERG, & JORGENSEN, 2001). Para a terceira etapa é utilizada a métrica de pontos por função (VAZQUEZ, SIMÕES, & ALBERT, 2003) e para a quarta etapa é utilizado o conjunto de métricas orientadas a objetos proposta por Chidamber e Kemerer (1994), conhecidas como métricas CK. A Tabela 4 apresenta as métricas utilizadas a cada etapa da abordagem.

Tabela 4 - Métricas de Complexidade

Etapa da Abordagem	Tipo de Requisito Funcional	Métrica de complexidade
1ª Etapa	Casos de uso	Pontos por caso de uso
2ª Etapa	Subcasos independentes	Pontos por caso de uso
3ª Etapa	Subcasos dependentes	Pontos por função
4ª Etapa	Serviços técnicos	Conjunto de métricas CK

Fonte: Adaptado de (PIMENTEL, 2007)

De acordo com (PIMENTEL, 2007), há duas principais razões para que estas métricas tenham sido adotadas. A primeira é que são bastante conhecidas no meio de desenvolvimento de *software*, o que facilita sua aplicação. A segunda é que tanto pontos por caso de uso e pontos por função, quanto o conjunto de métricas CK possuem formas bem estabelecidas, o que facilita o cálculo do conteúdo da informação do projeto em cada uma das etapas.

Os pontos por caso de uso são originados a partir da métrica de pontos por função. Estas métricas foram originalmente elaboradas para a definição do tamanho dos sistemas de *software* para a estimativa de esforço necessário para o desenvolvimento desse tipo de projeto. Os cálculo de pontos por caso de uso inicia-se pela avaliação da complexidade de cada ator e cada caso de uso e a atribuição de pesos a cada um deles. Estes pesos são somados e são obtidos o pontos de casos de uso não ajustados. A este valor são aplicados os fatores de ajuste técnico e ambiental, obtendo-se assim o valor para os pontos por caso de uso. Os fatores de ajuste técnico e de ambiente levam em consideração fatores como usabilidade, reusabilidade, complexidade de processamento, concorrência de processos, experiência da equipe, motivação, entre outros.

A métrica de pontos por função é uma métrica amplamente utilizada para a estimativa de tamanho de sistemas computacionais (PIMENTEL, 2007). A contagem de pontos por função permite que o tamanho do sistema seja calculado a partir dos seus requisitos funcionais. Nesta técnica as funções são classificadas em funções de dados e de transações. As classes definidas ainda podem ser divididas por níveis de complexidade. Assim, cada função recebe uma pontuação e, sobre o somatório desta pontuação, é aplicado um fator de ajuste, semelhante ao que ocorre com os pontos por caso de uso.

Finalmente, o conjunto de métricas CK utiliza características de classes individuais para realizar medições de complexidade. Como os requisitos funcionais desta etapa são serviços técnicos e os parâmetros de projeto são objetos, faz-se necessário usar uma métrica que contemple características de classes, atributos e métodos (PIMENTEL, 2007). Nesta métrica são definidas seis grandezas que serão medidas (CHIDAMBER; KEMERER, 1994): (1) métodos ponderados por classe; (2) profundidade da árvore de herança; (3) número de subclasses; (4) acoplamento entre objetos; (5) respostas para a classe e (6) falta de coesão nos métodos.

3.7 Conclusões

Este capítulo apresentou a Teoria de Projeto Axiomático descrevendo seus principais conceitos, técnicas e ferramentas com ênfase na sua aplicação ao desenvolvimento de *software* orientado a objetos. Foi apresentada uma introdução à teoria e suas definições básicas, os principais conceitos do Projeto Axiomático relacionados ao Axioma da Independência e conceitos relacionados com o Axioma da Informação. Além disso, foram apresentados alguns trabalhos relacionados com a Teoria de Projeto Axiomático e desenvolvimento de *software*. O capítulo também apresentou com a abordagem de projeto de *software* orientado a objetos baseada na Teoria de Projeto Axiomático, que sugere uma aplicação em conjunto com o Processo Unificado.

A aplicação em conjunto destas duas técnicas, pode trazer vantagens ao desenvolvimento de *software* orientado a objetos como critérios para a escolha da melhor solução de projeto para satisfazer um conjunto de requisitos funcionais, permitir ao projetista um maior detalhamento dos requisitos funcionais como a decomposição funcional e o processo em *zig-zag*, e oferecer um mecanismo de rastreabilidade dos componentes do sistema em relação aos requisitos dos quais foram originados.

Apesar da contribuição de Suh e seus colegas para esta discussão e do avanço oferecido pelo trabalho de Pimentel (2007) no desenvolvimento de *software* orientado a objetos, nenhum desses trabalhos foca a análise de negócio e especificação de requisitos. Conforme discutido no Capítulo anterior, as atividades de análise do problema e das necessidades dos envolvidos são fundamentais para uma especificação de requisitos de alta qualidade. Desta forma, uma abordagem de especificação de requisitos que possa ser integrada aos trabalhos destes autores pode ser uma contribuição de grande valor para a Engenharia de *Software*.

4 Especificação de Requisitos Baseada no Projeto Axiomático

Neste Capítulo é descrita a abordagem de especificação de requisitos proposta nessa dissertação. Esta abordagem aplica o Axioma da Independência ao processo de especificação de requisitos de sistemas de *software*. Introduce-se a abordagem proposta, apresentando seus objetivos, aplicações e vantagens. A seguir são estabelecidos os domínios de projeto de sistemas de *software* e seus relacionamentos. O passo seguinte é a definição das correspondências entre os domínios de Projeto Axiomático (PA) e o as fases de um processo de *software*, no caso o Processo Unificado (PU). A próxima seção discute uma forma de manter a correspondência entre elementos dos diferentes domínios nos diferentes níveis de detalhamento. Em seguida é discutida a rastreabilidade bidirecional de requisitos na abordagem proposta por meio das matrizes do PA. A seção seguinte apresenta os conceitos utilizados em quatro etapas de decomposição dos elementos pertencentes aos domínios do problema, do cliente e funcional. Na mesma seção é estabelecida uma forma para se representar estes conceitos utilizando linguagens de modelagem como a UML e a SysML. Em seguida são descritas as etapas da abordagem proposta e as atividades realizadas nestas etapas. Finalmente, é descrito como o processo de *zigzagamento* do PA é adaptado para as atividades de especificação de requisitos nas quatro etapas de decomposição.

4.1 Introdução à Abordagem Proposta

O principal objetivo deste trabalho é propor uma abordagem que permita aplicar a Teoria de Projeto Axiomático (TPA) (SUH, 1990) na especificação de requisitos de sistemas de *software*. Outro objetivo é integrar a abordagem proposta a um processo estabelecido de especificação de requisitos como o fluxo de requisitos do Processo Unificado (JACOBSON, BOOCH, & RUMBAUGH, 1999). Desta forma, esta abordagem de especificação de requisitos poderá ser integrada aos processos e métodos de desenvolvimento de *software* usados atualmente. Da mesma forma, a abordagem proposta poderá ser integrada à abordagem de projeto orientado a objetos baseado em projeto axiomático (PIMENTEL, 2007).

A criação de uma abordagem que aplica a Teoria de Projeto Axiomático à especificação de requisitos de *software* em conjunto com o Processo Unificado se propõe a trazer vantagens ao processo de desenvolvimento de sistemas de *software*. Entre estas vantagens estão: estabelecer um método para a análise dos problemas e do domínio do problema; estabelecer um método que facilite a identificação e entendimento das necessidades do cliente; e rastrear os requisitos deste de sua origem, dos problemas do cliente até os casos de uso.

As metodologias de especificação de requisitos tradicionais utilizam recursos como entrevistas, *workshops* de requisitos, *brainstorming* e *storyboards* (LEFFINGWELL & WIDRIG, 2003) para entender os problemas e necessidades do cliente. Assim, frequentemente são definidos os requisitos do sistema sem que seja verificado se os problemas e necessidades definidos pelo cliente estão claros ou se existem questões que ele não deixou explícitas. Normalmente, as informações fornecidas pelos clientes são ambíguas e obscuras. Caso não sejam analisadas com cuidado levarão à definição de um conjunto inadequado de requisitos ou à definição de soluções de projeto ineficazes.

Desta forma, as abordagens de especificação de requisitos não propõem a decomposição de problemas e necessidades do cliente. A maioria delas tampouco sugere o detalhamento de requisitos. Nestas abordagens, estes elementos costumam ser interpretados apenas em um nível alto de abstração, o que pode tornar a sua análise superficial. Diferentemente, algumas abordagens, ditas baseadas em problemas, como a proposta por (JACKSON, 1999), defendem a importância de se analisar o problema de maneira detalhada com o objetivo de simplificar a elaboração da arquitetura do sistema.

Por outro lado, a TPA propõe a decomposição dos requisitos funcionais em uma estrutura hierárquica detalhada (SUH, 2001). Porém, não faz nenhuma menção à decomposição das necessidades do cliente, apenas apresenta-as como uma fonte para a identificação dos requisitos funcionais. Além disso, a TPA não define o domínio do problema e, portanto, também não trata a decomposição de problemas.

Na abordagem desenvolvida, em primeiro lugar, é proposta uma mudança no conjunto de domínios da TPA, adicionando o Domínio do Problema ao conjunto de domínios de projeto definido por Suh (1990). Esta nova visão permite incluir as tarefas de análise do problema na especificação de requisitos proposta por meio desta abordagem. Tendo em vista essa mudança na visão de domínios, a

abordagem redefine a correspondência entre os domínios da Teoria de Projeto Axiomático (TPA) e os fluxos do Processo Unificado (PU), com foco na relevância de cada fluxo de acordo com as fases de projeto. Esta abordagem também estabelece os conceitos que são aplicados em cada nível de detalhamento dos Problemas, Necessidades e Requisitos. Além disso, a abordagem apresenta elementos de modelagem que representam estes conceitos em linguagem UML/SysML. Finalmente, define as etapas e atividades do modelo de especificação de requisitos da abordagem.

De maneira geral, o trabalho propõe o uso do Axioma da Independência (AI) para avaliar se as necessidades são independentes entre si, em relação aos problemas que ajudam a resolver. Da mesma maneira, o AI é utilizado para avaliar a independência dos requisitos em relação às necessidades que atendem. Desta forma, é possível identificar os vínculos existentes entre estes elementos. Por outro lado, a decomposição dos mesmos elementos em vários níveis de hierarquia tem papel fundamental na compreensão dos problemas e necessidades. Por meio destes dois mecanismos se torna possível identificar e resolver inconsistências na análise e, conseqüentemente, definir com mais segurança os requisitos do sistema.

Além disso, o uso das ferramentas de Projeto Axiomático, como a matriz de projeto (MP) e a hierarquia funcional (processo em *zig-zag*), facilita a rastreabilidade dos requisitos em relação às necessidades e problemas (elemento incluído por esta abordagem). Para cada Problema (PB) é identificado o conjunto das Necessidades do Cliente (NC) vinculadas. Assim, cada PB é relacionado com as NCs correspondentes. Esta correspondência é mapeada na matriz de projeto.

Da mesma forma, para cada NC são identificados os Requisitos (RQs) que a atendem, sendo que cada NC é relacionada com os RQs correspondentes na matriz de projeto. Por sua vez, cada RQ de mais baixo nível está relacionado aos elementos de mais alto nível de abstração por meio da hierarquia funcional. O mesmo conceito é válido para os PBs e NCs. Este mecanismo permite uma rastreabilidade tanto horizontal (ex: de necessidades para problemas) quanto vertical (ex: de requisitos para sub-requisitos). Desta forma, é possível que um componente de *software* seja rastreado até o requisito de alto nível que satisfaz e para problema que originou este requisito.

A capacidade de rastrear requisitos é um meio comprovadamente eficaz para aumentar a qualidade e a confiabilidade do *software*, bem como para garantir

que os requisitos do sistema foram atendidos. Além disso, é muito importante para a minimização de impacto de mudanças de requisitos (LEFFINGWELL & WIDRIG, 2003). Esta capacidade é exigida em modelos de maturidade de *software* como MPS.BR (Softex, 2011) e CMMI (CHRISISS, KONRAD, & SHRUM, 2007). Estes modelos de maturidade têm como objetivo (ou exigência) alcançar e manter a rastreabilidade dos requisitos em relação aos modelos de arquitetura e códigos fonte.

A abordagem proposta nesta dissertação pode ajudar a atingir um nível suficiente de rastreabilidade por meio da aplicação dos axiomas, teoremas, ferramentas e processos da TPA. Além disso, pode ser um método eficiente para se alcançar um alto grau de entendimento dos requisitos que o sistema deve atender.

4.2 Domínios do Desenvolvimento de Sistemas de *Software*

De maneira geral, pode-se classificar os domínios de desenvolvimento de sistemas de *software* em dois grupos: Domínio de Problema e Domínio da Solução. Leffingwell e Widrig apresentam uma divisão destes dois domínios em alguns níveis, como ilustra a Figura 26 (LEFFINGWELL & WIDRIG, 2003). De acordo com os autores, a nuvem representa o conhecimento confuso e desorganizado que compõe o domínio do problema. As necessidades dos *stakeholders* pertencem também ao mesmo domínio. Por outro lado, no domínio da solução encontram-se as características e os requisitos do *software*.

O domínio do problema costuma ser um dos maiores desafios dos analistas de negócio e de requisitos. Entender o mundo do cliente, seus termos técnicos, regras e processos de negócio, além dos problemas que precisam ser resolvidos, pode se tornar um processo longo e complicado. O usuário, em geral, tende a ser um completo estranho que vive em um mundo desconhecido para os desenvolvedores.

Esse usuário precisa de ajuda para resolver seus problemas técnicos e, portanto, se torna responsabilidade da equipe de desenvolvimento entender os problemas dele. Nessa situação, tratar o domínio do problema como uma “nuvem” de informações confusas e desorganizadas pode dificultar ainda mais o trabalho da

equipe. Nesse contexto, é necessário que a equipe domine técnicas de análise do problema e especificação de requisitos.



Figura 26 - Visão de Domínios de Problema e Solução
Fonte: (LEFFINGWELL & WIDRIG, 2003)

Assim sendo, o domínio do problema foi reestruturado, com objetivo de organizar a “nuvem” em novos níveis da pirâmide como ilustra a Figura 27. Conforme a visão proposta neste trabalho, o domínio do problema passa a ser organizado em três níveis da pirâmide: Necessidades Humanas ou Organizacionais, Problemas e Necessidades do Cliente. Por outro lado, no o domínio da solução é organizado em outras três partes: Características da Solução, Requisitos do Sistema e Casos de Uso.

Aqui é importante salientar que, embora os termos utilizados sejam semelhantes, o conceito de Necessidades Humanas ou Organizacionais é completamente diferente do conceito de Necessidades do Cliente. Como discutido na seção 2.4, as Necessidades Humanas correspondem ao conjunto de necessidades que são percebidas pelas pessoas ou instituições de maneira geral. As Necessidades do Cliente, por sua vez, referem-se somente às necessidades que o cliente espera que sejam atendidas por um determinado sistema.

De maneira geral, o relacionamento que existe entre os níveis da pirâmide é de causa e efeito. Assim sendo, são as necessidades básicas que, quando em desequilíbrio, causam os problemas. Em consequência, a existência dos problemas provoca as necessidades, que pedem por uma ou mais soluções. Finalmente, são as necessidades dos clientes que definem quais requisitos são exigidos de um produto (de forma que o sistema ajude a solucionar os problemas). Assim sendo,

partindo dos problemas e passando por cada um dos níveis da pirâmide, é possível para o analista sistematizar a especificação dos requisitos. Para identificar estas relações pode-se simplesmente perguntar “**quais** necessidades este problema gera?” ou “**como** é possível resolver este problema?”.



Figura 27 - Nova Visão de Domínios de Problema e Solução
Fonte: Autoria Própria

Da mesma forma, partindo dos requisitos e perguntando “por quê?” os analistas serão capazes de chegar às necessidades. Esta rastreabilidade horizontal que deve existir entre os domínios é viabilizada pelo uso de matrizes. Nestas matrizes relacionam-se os elementos de cada domínio com os elementos do domínio seguinte, da esquerda para a direita. Além disso, nesta abordagem a matriz resultante deste relacionamento será avaliada segundo o Axioma da Independência conforme discutido mais adiante neste capítulo. A dinâmica desta relação é ilustrada na Figura 28.

Para efeito da abordagem proposta de especificação de requisitos não é considerado o primeiro nível, das Necessidades Humanas e Organizacionais. Especialmente por que isso levaria a um estudo subjetivo das razões que levam o cliente a querer resolver determinado problema. Este tipo de análise foge do contexto técnico da análise de requisitos, embora possa ser favorável ao analista ter certa sensibilidade em relação às motivações e expectativas do cliente.

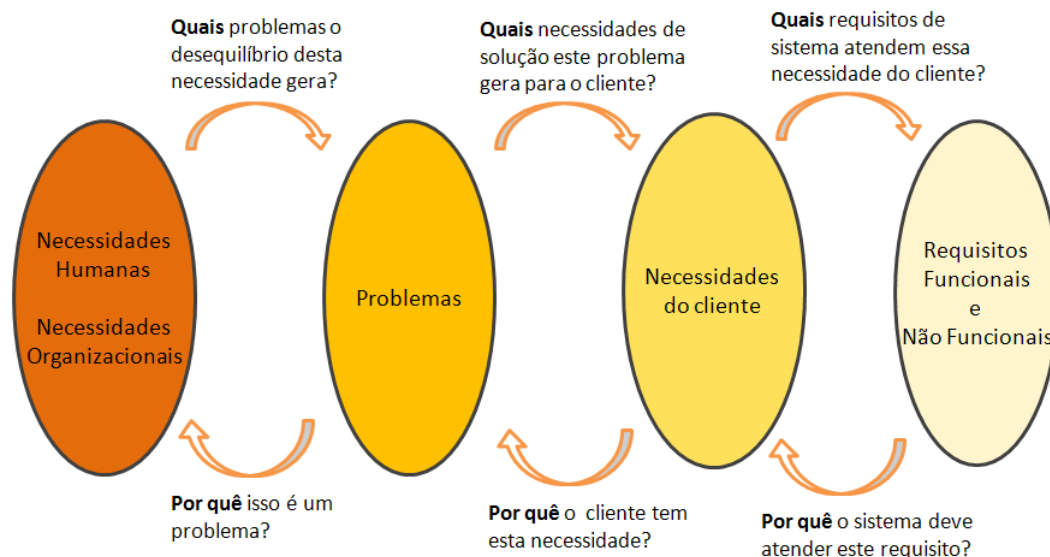


Figura 28 - Relacionamento Entre Domínios
Fonte: A autoria Própria

Assim sendo, esta abordagem deixa de lado a questão das necessidades humanas/organizacionais, porém acrescenta um novo domínio ao conjunto de domínios de projeto propostos por Suh (SUH, 2001) como ilustra a Figura 29. Este novo domínio, denominado Domínio do Problema, contém toda a informação que define o problema e leva ao surgimento das necessidades do cliente. Por sua vez, as características se juntam às necessidades do cliente no domínio do cliente. Essa junção se deve ao fato de que as características representam descrições, na linguagem do cliente, do que o sistema precisa fazer para atender as necessidades deste cliente. Portanto, nesta abordagem as características são consideradas parte do domínio do cliente, por estarem relacionadas de uma maneira muito próxima com as necessidades do cliente (LEFFINGWELL & WIDRIG, 2003).

A proposta deste trabalho é iniciar o processo de especificação de requisitos pela identificação de Problemas e seu detalhamento. Em seguida, passa-se ao mapeamento dos Problemas para as Necessidades e o detalhamento das necessidades. Este processo de mapeamento e detalhamento entre domínios segue para o domínio funcional, chegando à identificação do conjunto dos requisitos funcionais e não funcionais. O conjunto dos requisitos definidos origina os casos de uso e, posteriormente, a arquitetura do produto.

Assim, tem-se nesta abordagem a representação do Domínio Funcional em duas partes, conforme representado pela Figura 29. A primeira, situada na área de Engenharia de Sistema, contém os Requisitos Funcionais e Não Funcionais de

Sistema. A segunda pertence à área de Engenharia de *Software* e contém os requisitos de *software*, representados por Casos de Uso. Esta organização é importante para que a abordagem de especificação de requisitos proposta neste trabalho possa ser utilizada em conjunto com a abordagem de Pimentel (2007).

Além disso, esta divisão do domínio funcional é importante pelo fato de se considerar, neste trabalho, que a elicitação de requisitos pertence à Engenharia de Sistemas. Considera-se, assim, que após a sua especificação alguns requisitos de sistema serão transferidos para a Engenharia de *Software* (transformando-se em casos de uso), enquanto outros poderão ser transferidos para áreas como Eletrônica ou Mecânica, por exemplo. Essa visão permite que a abordagem de especificação de requisitos proposta seja utilizada inclusive em áreas do conhecimento não necessariamente relacionadas com o desenvolvimento de *software*.

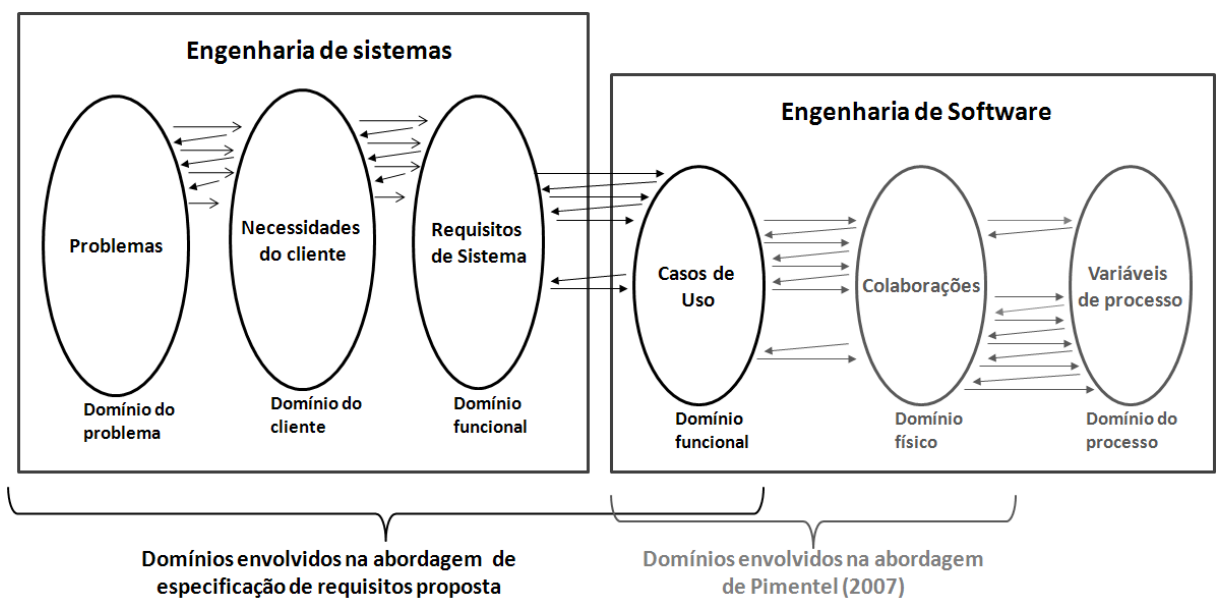


Figura 29 - Domínios de projeto
Fonte: Autoria própria

Uma vez definidos os domínios envolvidos na especificação de requisitos, faz-se necessário determinar a forma como a abordagem proposta é aplicada no desenvolvimento de sistemas de *software*. A maneira mais adequada encontrada para a aplicação da abordagem foi adaptá-la a processos, até então, consolidados no meio de desenvolvimento, como o Processo Unificado.

4.3 Projeto Axiomático e Processo Unificado

Conforme apresentado no trabalho de Pimentel (2007) é possível estabelecer uma relação entre as fases do Processo Unificado (PU) e os domínios do Projeto Axiomático (PA). Esse trabalho propôs que os processos de mapeamento entre os quatro domínios definidos por Suh (2001) fossem relacionados às quatro fases do PU, como visto no Capítulo 3.

Neste trabalho, apresenta-se uma releitura desta visão, que se tornou possível devido à inclusão do domínio do problema no conjunto de domínios do PA. Assim, a abordagem proposta estabelece uma relação do processo de mapeamento entre os domínios com os fluxos do PU ligados mais diretamente ao desenvolvimento de *software* (modelagem de negócio, especificação de requisitos, modelagem e *implementação*). Por sua vez, é a intensidade com que ocorre o mapeamento entre determinado par de domínios a cada fase que relaciona as fases PU à abordagem proposta. O relacionamento estabelecido entre a abordagem proposta e o Processo Unificado é ilustrado pela Figura 30.

Assim, como se pode observar na Figura 30, o processo de mapeamento entre problemas e necessidades corresponde à disciplina de Modelagem de negócio do Processo Unificado. O mapeamento entre necessidades e requisitos corresponde à disciplina de Requisitos, enquanto o mapeamento de requisitos para elementos da UML no domínio físico corresponde à disciplina de Análise e *Design*.

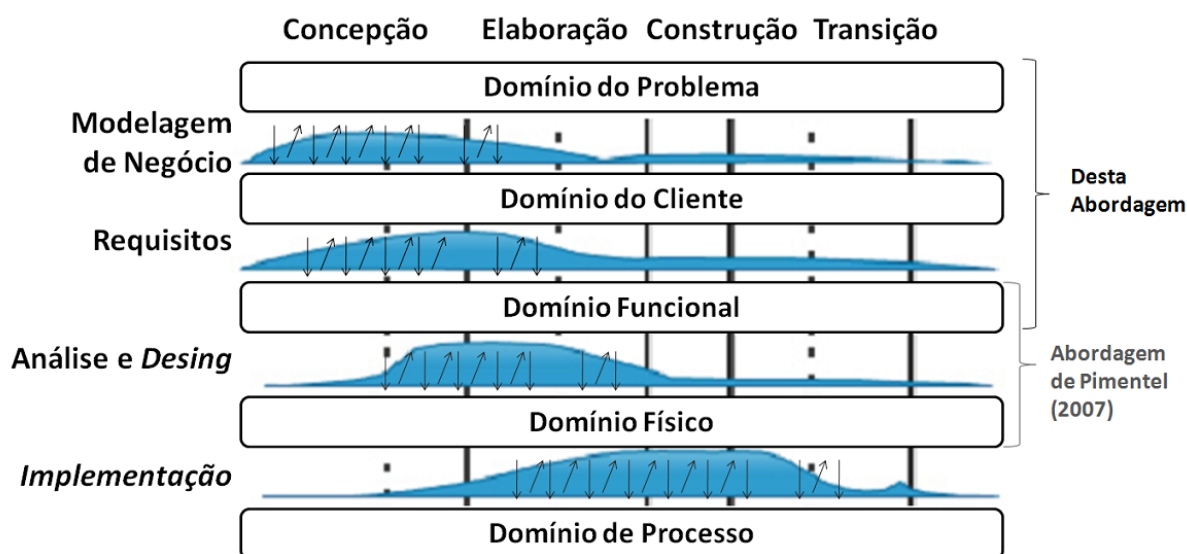


Figura 30 - Projeto Axiomático e fases do Projeto Unificado
Fonte: Autoria própria

Desta forma, trabalha-se no relacionamento entre cada domínio com mais ou menos intensidade conforme a fase do processo de desenvolvimento. Por exemplo, na fase de concepção o foco está, principalmente, no mapeamento entre os domínios do problema, do cliente e funcional, pois nesta fase são estudados os modelos de negócios, as necessidades do cliente e os requisitos do sistema. Por outro lado, na fase de elaboração o foco está no relacionamento entre domínio funcional e físico à medida que estão sendo construídos os modelos do sistema.

Nas seções anteriores falou-se do mapeamento de elementos de um domínio para outro, seguido pela decomposição dos elementos destes domínios. Essa descrição do processo pode fazer parecer que ele envolve somente dois domínios. No entanto, é necessário esclarecer que o processo de mapeamento não deve ser independente, mas sim, um processo completo, que pode envolver três ou mais domínios a cada iteração como ilustra a Figura 30. Mais adiante neste Capítulo é discutido o estabelecimento de etapas de decomposição dos elementos dos domínios e o foco de cada uma delas, levando em conta a relação estabelecida com o PU nesta Seção.

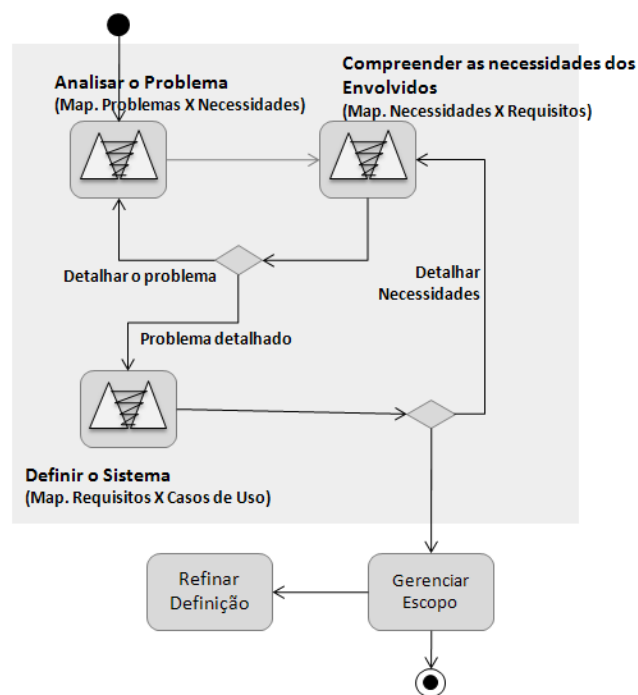


Figura 31 - Fluxo de Requisitos do PU e Abordagem Proposta
 Fonte: Autoria própria

Uma vez que o trabalho apresentado nessa dissertação está focado, principalmente, na especificação de requisitos, buscou-se estabelecer também a relação da abordagem proposta com o fluxo de requisitos do PU. Esta relação é ilustrada na Figura 31. Desta forma, a atividade “Analisar Problema” do PU pode ser representada na abordagem proposta como a atividade de “identificação de problemas e seu mapeamento para as necessidades”. A atividade “Compreender as necessidades dos envolvidos” toma, nesta abordagem, a forma da “identificação das necessidades e seu mapeamento para os requisitos”. Finalmente, a atividade “Definir Sistema” é representada pelo “mapeamento dos requisitos para casos de uso do *software*”. Por sua vez, as atividades de Gerenciar escopo e Refinar a Definição do Sistema não fazem parte do escopo da abordagem proposta, como ilustra a Figura 31.

Esta seção estabeleceu a relação existente entre os domínios do Projeto Axiomático e as fases do Processo Unificado. Desta forma, de acordo com a fase do projeto, pode-se ter mais foco no mapeamento entre um grupo ou outro de domínios. Além disso, discutiu-se a relação existente entre o fluxo de requisitos e a abordagem proposta. A próxima seção discute a questão da manutenção da correspondência entre elementos destes domínios do ponto de vista do nível de detalhamento que se pode alcançar em cada um deles.

4.3.1 Mapeamento e Níveis de Detalhamento

Conforme discutido anteriormente, a atividade de representar o mapeamento dos relacionamentos existentes entre elementos dos domínios funcional e físico em uma matriz é uma das características fundamentais da TPA. Ainda, o número de elementos deve ser sempre igual nos dois domínios possibilitando a formação de uma matriz quadrada em qualquer nível de decomposição funcional, segundo o Teorema 4 (Anexo A).

No entanto, este trabalho tem por objetivo estender o processo de mapeamento e decomposição para os domínios do problema e do cliente. Portanto, embora seja desejável manter sempre o mesmo número de elementos em todos os domínios, nesta abordagem torna-se comum um domínio possuir mais itens que o anterior, pois quanto mais se caminha em direção ao domínio físico maior o nível de

detalhamento que pode ser obtido. Ou seja, quanto mais o processo se aproxima da *implementação*, maior tende a ser o detalhamento do projeto. Sendo que, em última análise, no nível máximo de abstração de um projeto de *software* encontram-se os problemas do cliente enquanto no nível máximo de detalhamento estão as linhas do código fonte.

Assim sendo, recomenda-se manter o mapeamento entre dois domínios até que não seja mais possível ou interessante realizar detalhamentos. A partir do momento em que não for mais possível detalhar, deixa-se de realizar o mapeamento entre os domínios envolvidos. Daí em diante, passa-se a fazer mapeamento somente entre os outros domínios em que ainda é possível continuar realizando decomposição.

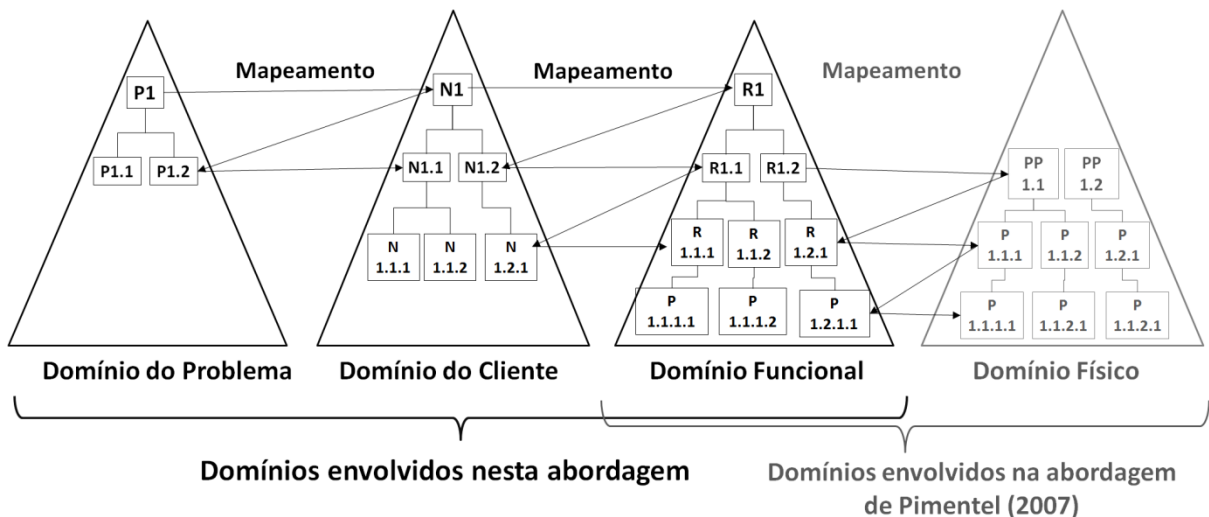


Figura 32 - Níveis de Detalhamento
Fonte: Autoria própria

Ao final do projeto, um domínio terá mais níveis de detalhamento que o outro, porém, até o nível de decomposição alcançado, existirá sempre o mesmo número de elementos entre os pares de os domínios, conforme ilustra a Figura 32. Por outro lado, é possível que alguns domínios não possuam elementos nos níveis mais altos de abstração, conforme ilustrado na figura pelo Domínio Físico, que, como se pode ver, não possui nenhum elemento correspondente ao requisito “R1”. Isso se deve ao fato de que alguns domínios, como o físico, são mais voltados aos aspectos práticos e, portanto, têm limitações em relação à abstração.

Esta seção discutiu a questão da manutenção da correspondência entre elementos dos domínios do ponto de vista do nível de detalhamento que se pode

alcançar em cada um deles. Ainda levando em conta a questão do detalhamento e mapeamento dos elementos de cada domínio, a próxima seção discute as questões relativas à rastreabilidade que pode ser obtida com a aplicação da abordagem proposta.

4.4 Projeto Axiomático e Rastreabilidade de Requisitos

A rastreabilidade de requisitos é uma prática importante no processo de desenvolvimento de *software* e é uma exigência de modelos de maturidade como CMMI (CHRISSIS, KONRAD, & SHRUM, 2007) e MPSBR (Softex, 2011). Esta prática tem por objetivo manter a rastreabilidade bidirecional dos requisitos do projeto (CHRISSIS, KONRAD, & SHRUM, 2007). Assim, rastreabilidade deve cobrir tanto os relacionamentos horizontais, estudando o impacto que os requisitos têm uns sobre os outros, quanto os verticais, rastreando os relacionamentos entre os requisitos em seus níveis de decomposição.

Quando os requisitos são bem gerenciados, a rastreabilidade pode ser estabelecida, desde um requisito fonte até seus requisitos de mais baixo nível e destes de volta para o seu requisito fonte. Essa rastreabilidade, dita vertical, auxilia a determinar se todos os requisitos fonte foram completamente tratados. Da mesma forma, auxilia a verificar se todos os requisitos de mais baixo nível podem ser rastreados para uma fonte válida. Exceção deve ser feita aos requisitos ditos “de infra-estrutura” que não possuem requisitos fonte uma vez que eles derivam de necessidades da arquitetura da solução e não de outros requisitos ou necessidades.

Além disso, a rastreabilidade de requisitos permite analisar o impacto que uma alteração de requisito, por exemplo, terá sobre os demais requisitos do sistema. Da mesma maneira, esta rastreabilidade horizontal permite analisar os relacionamentos que um elemento do domínio físico possui com elementos do domínio funcional e entender o impacto que uma alteração de requisito terá no domínio físico.

A matriz de projeto, se utilizada conforme a abordagem de Pimentel (2007), permite identificar as dependências entre RFs e PPs, facilitando o rastreamento horizontal dos requisitos. Por outro lado, a matriz permite uma rastreabilidade vertical ao relacionar os requisitos com os níveis mais baixos da hierarquia funcional

até chegar às classes. Desta maneira, pode-se considerar que a matriz de projeto axiomático é equivalente à matriz de rastreabilidade dos requisitos.

Em seu livro, Leffingwell sugere a aplicação de matrizes de rastreabilidade entre os diferentes domínios de projeto e exemplifica com uma matriz que mapeia Características para Necessidades (LEFFINGWELL & WIDRIG, 2003). Seguindo a proposta destes autores, neste trabalho utilizam-se as matrizes de rastreabilidade para mapear as relações existentes entre os diferentes domínios de projeto. Adicionalmente, deve ser verificada a conformidade de cada matriz obtida com o Primeiro Axioma. Esta verificação é realizada para garantir a independência da solução desde as fases iniciais do projeto, com objetivo de garantir que o resultado final do projeto seja um sistema de alto nível de qualidade e que satisfaça as necessidades do cliente.

Nesta abordagem a rastreabilidade vertical refere-se ao relacionamento entre elementos de um mesmo domínio, nos diferentes níveis de abstração. Por sua vez, a rastreabilidade horizontal refere-se ao relacionamento entre elementos de diferentes domínios em um mesmo nível de abstração. A mesma visão pode ser estendida a todos os domínios do desenvolvimento de sistemas. A rastreabilidade bidirecional dos elementos que compõem a especificação de requisitos é ilustrada na Figura 33.

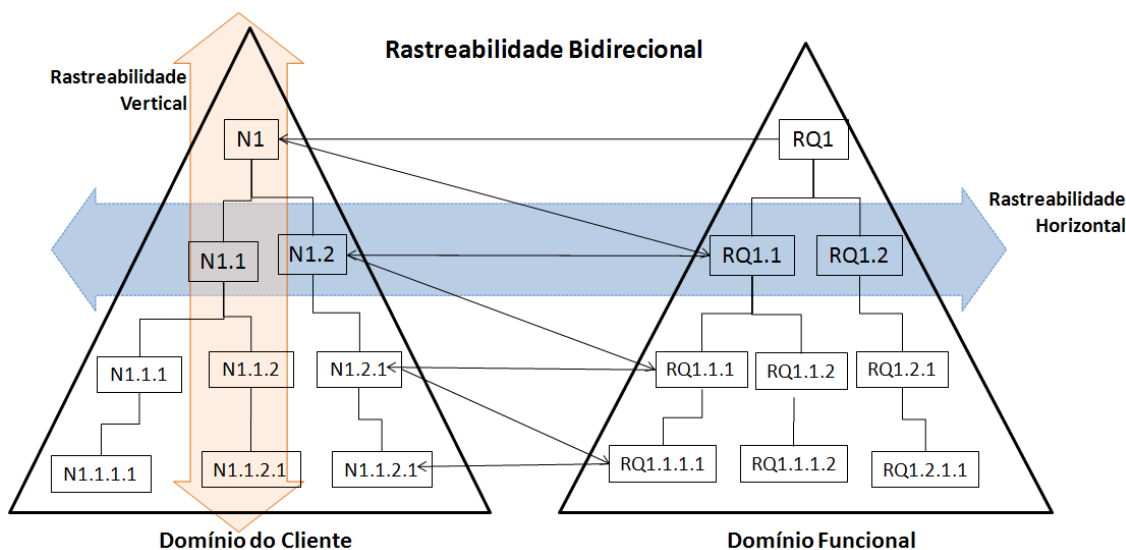


Figura 33 - Rastreabilidade Bidirecional
Fonte: Autoria Própria

A título de ilustração, a Figura 34 traz um exemplo de matriz de rastreabilidade de um projeto de sapato. Esta matriz relaciona os Requisitos com as Necessidades que cada um atende. De acordo com o cliente, algumas de suas necessidades são “Reduzir Dores nos Joelhos” e “Reduzir Dores nos Dedos”. Para estas necessidades foram definidas os requisitos “Proteção dos Joelhos Contra Impactos” e “Proteção para os Dedos”. Como pode ser observado na figura, a matriz relaciona tanto elementos de diferentes domínios, as necessidades e os requisitos, quanto elementos de um mesmo domínio em diferentes níveis da hierarquia funcional, requisitos (3) e sub-requisitos (R3.1, R3.2 e R3.3).

	Rq::Proteger Joelhos de Impactos	Rq::Reduzir Cansaço das Pernas	Rq::Reduzir Machucaduras nos Pés	Rq::Reduzir dores nas Planta do Pé	Rq::Reduzier Dores no Calcanhar	Rq::Evitar Dores nos Dedos
UN::MecanismoReducaoDoresNosJoelhos	X					
UN::MecanismoReduzirDoresNasPernas		X				
UN::MecanismoReduzirMachucaduras			X			
UN::MecanismoProtecaoPlantar				X		
UN::MecanismoProtecaoDedos					X	
UN::MecanismoProtecaoCalcanhar						X

Figura 34 - Matriz de Necessidades X Requisitos
 Fonte: Autoria Própria

Esta seção discutiu a importância das matrizes de projeto axiomático para a rastreabilidade de requisitos. Encerra-se, assim, a contextualização da abordagem proposta em relação à engenharia de *software* e de requisitos. A próxima seção apresenta os conceitos empregados nesta abordagem e os elementos de modelagem visual utilizados na sua aplicação.

4.5 Conceitos Empregados na Abordagem Proposta

Conforme discutido previamente, no trabalho de Suh (2001) é proposta a realização de um processo de mapeamento e decomposição dos elementos do domínio funcional e do domínio físico, denominado processo em *zig-zag*. Esse processo se caracteriza por alternar as atividades de mapeamento e detalhamento de problemas e necessidades. Por sua vez, Pimentel (2007) dá um novo passo ao definir os elementos que representam Requisitos Funcionais e Parâmetros de Projeto em diferentes níveis de abstração do processo de *zig-zagueamento*.

Neste trabalho sugere-se estender o processo em *zig-zag* aos Domínios do Problema e do Cliente e, portanto, é necessário estabelecer conceitos para representar os diferentes níveis de abstração na decomposição dos elementos dos domínios envolvidos. A Figura 35 ilustra a realização do processo em *zig-zag* entre dois domínios de forma que seus elementos são representados por diferentes conceitos definidos em cada uma das etapas de decomposição.

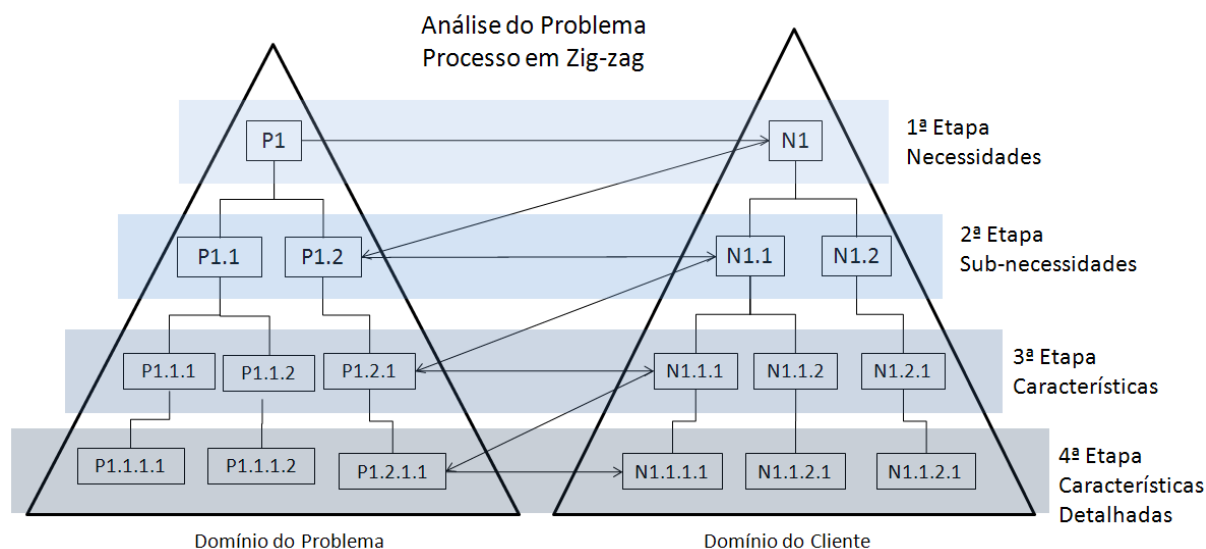


Figura 35 - Análise do Problema
Fonte: Autoria própria

A abordagem proposta nesta dissertação define quatro níveis de abstração para os elementos pertencentes aos domínios do problema, do cliente e funcional. Estes níveis de abstração servem de base para o processo de decomposição realizado neste modelo e para a definição das etapas da abordagem. É importante lembrar que estes elementos possuem correspondentes nos Domínios Funcional e

Físico conforme definido por Pimentel (2007) e apresentado no Capítulo 3 desta dissertação. A Tabela 5 apresenta um quadro que relaciona os elementos de cada domínio de acordo com o nível de abstração do projeto e seus correspondentes nos demais domínios.

O estabelecimento destes conceitos permite ainda que seja estabelecida uma linguagem para a modelagem visual destes elementos. A UML, embora seja uma das linguagens de modelagem mais utilizadas na atualidade, está focada principalmente no aspecto funcional e não define elementos para representar requisitos não funcionais. Além disso, a UML não possui elementos que representem Problemas e Necessidades. Portanto, é uma linguagem limitada ao desenvolvimento de *software* e que não serve aos objetivos da análise de problemas e necessidades. Por outro lado, a SysML, que foi definida por meio de um perfil UML (WEILKIENS, 2007), possui alguns elementos e diagramas que se aproximam dos conceitos definidos neste trabalho. O principal deles é o conceito de requisitos e a elaboração de diagramas de requisitos. No entanto, a maior parte dos conceitos desta abordagem não encontram correspondentes na UML ou SysML. Portanto, foi necessário fazer uso do mecanismo de extensão, mais especificamente estereótipos, para completar o conjunto de conceitos da abordagem.

Tabela 5 - Níveis de Abstração

Nível de abstração	Domínio do Problema	Domínio do Cliente	Domínio Funcional
1	Problemas	Necessidades	Requisitos Funcionais e Não Funcionais
2	Subproblemas	Subnecessidades	Sub-requisitos Funcionais e Não Funcionais
3	Subproblemas	Características	Sub-requisitos Essenciais Funcionais e Não Funcionais
4	Subproblemas	Características Detalhadas	Sub-requisitos Técnicos Funcionais e Não Funcionais

Fonte: A autoria Própria

A seguir são definidos os conceitos correspondentes a cada um dos elementos utilizados em cada nível de abstração dos diferentes domínios da especificação de requisitos. Estes conceitos estão baseados nas discussões desenvolvidas na fundamentação teórica deste trabalho, na Seção 2.4. Além disso,

são identificados elementos de modelagem visual (UML/SysML) eventualmente empregados para cada um destes conceitos.

Problema: é algo que impulsiona o cliente buscar uma solução, gera uma necessidade. Para representar este elemento é necessário utilizar o estereótipo «*problem*» ou «problema» em uma classe da UML.

Subproblema: é uma decomposição de um problema, que auxilia na sua compreensão. Para representar este elemento é necessário utilizar o estereótipo «*subproblem*» ou «subproblema» em uma classe da UML.

Necessidade: é um comportamento do sistema que, na interpretação do cliente, irá resolver ou aperfeiçoar seu problema. Para representar este elemento é necessário utilizar o estereótipo «*need*» ou «necessidade» em uma classe da UML.

Subnecessidade: é uma decomposição de uma necessidade, que facilita seu entendimento. Para representar este elemento é necessário utilizar o estereótipo «*subneed*» ou «subnecessidade» em uma classe da UML.

Característica: é um atributo ou aspecto antecipado para atender uma necessidade do cliente. Para representar este elemento utiliza-se o estereótipo «*feature*» ou «característica» em uma classe da UML.

Característica Detalhada: é a decomposição de uma característica, que facilita o entendimento de seus detalhes. Para representar este elemento utiliza-se o estereótipo «*subfeature*» ou «subcaracterística» em uma classe da UML.

Requisito Funcional de Sistema: uma capacidade do *software* que o usuário necessita para resolver um problema, alcançar um objetivo ou que existe para satisfazer um contrato, padrão, especificação ou outra documentação imposta formalmente. A SysML define elementos para a representação de requisitos que podem receber o estereótipo «*functional*» para diferenciar-se dos requisitos não funcionais. Este mesmo elemento é utilizado nesta abordagem para representar todos os conceitos do domínio funcional.

Requisito Não Funcional de Sistema: descrevem restrições e qualidades de um sistema. Podem ser representados pelo mesmo elemento utilizado para representar os RFs e receber o estereótipo «*non-functional*» para diferenciá-los.

Sub-requisito de Sistema: é uma decomposição de um requisito, que facilita seu entendimento. Para representar utiliza-se o mesmo estereótipo dos requisitos.

Requisito Essencial de Sistema: requisito específico do domínio e independente de solução técnica (WEILKIENS, 2007). No processo de mapeamento entre Requisitos de Sistema e Requisitos de Software os requisitos essenciais são os correspondentes dos Casos de Uso Independentes de Características Técnicas (PIMENTEL, 2007).

Requisito Técnico de Sistema: requisito baseado em uma forma de solução (WEILKIENS, 2007). No processo de mapeamento entre Requisitos de Sistema e Requisitos de Software os requisitos técnicos são os correspondentes dos Casos de Uso Dependentes de Características Técnicas (PIMENTEL, 2007).

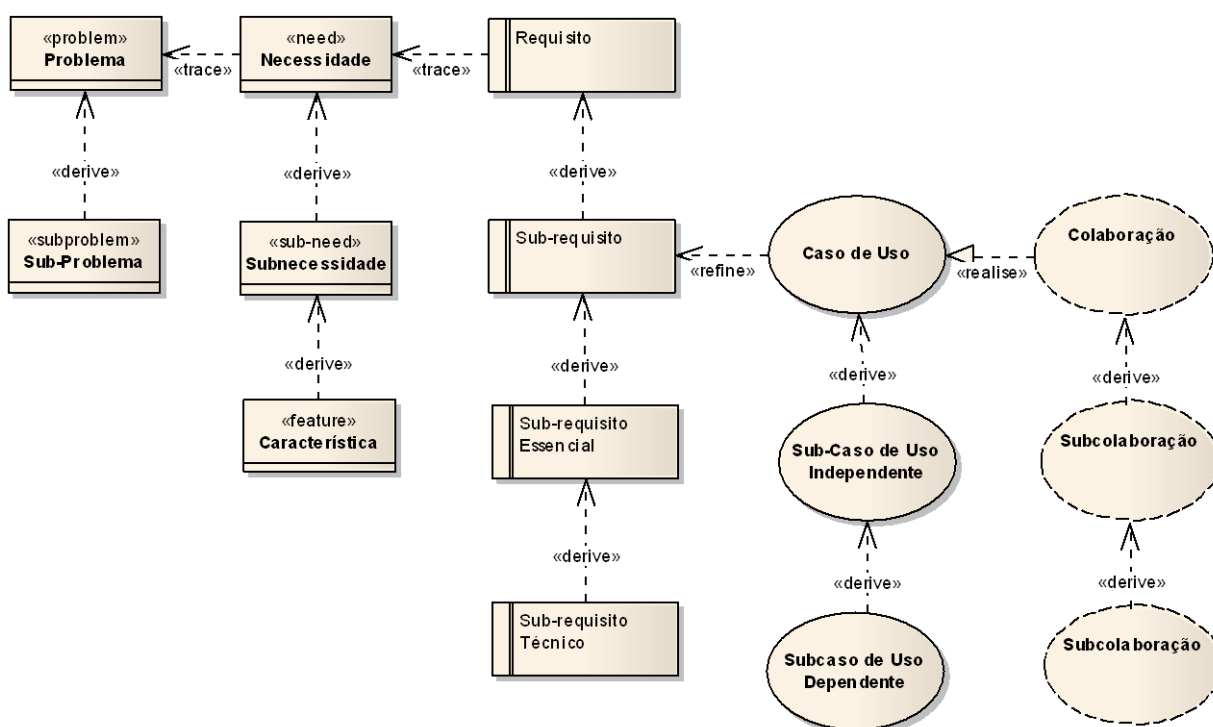


Figura 36 - Elementos UML Representando Elementos dos Domínios
 Fonte: Autoria Própria

Os elementos de modelagem pertencentes a cada um dos domínios de projeto e seus níveis de hierarquia são ilustrados na Figura 36. A definição destes elementos é importante no caso dos analistas de requisitos utilizarem ferramentas de modelagem visual para elaborar o sistema. Desta maneira, pode-se documentar o projeto de forma completa, desde a definição dos problemas, chegando até a

modelagem da arquitetura. Algumas ferramentas permitem, inclusive, gerar matrizes de rastreabilidade destes elementos. No entanto, não existe até o momento nenhuma destas ferramentas que verifique o nível de acoplamento destas matrizes. Uma iniciativa para verificar o nível de acoplamento destas matrizes é apresentada no Apêndice A deste trabalho.

Além dos elementos utilizados em cada domínio para representar os diferentes níveis de abstração, é necessário definir os tipos de relacionamentos que existem entre eles. Estes relacionamentos podem ser representado seguindo os padrões utilizados atualmente na UML e na SysML. A seguir são apresentadas as definições para alguns dos relacionamentos mais importantes para a modelagem de sistemas.

Relacionamento de Derivação de Requisitos: descreve que um requisito foi derivado de outro requisito (WEILKIENS, 2007). É representado pelo estereótipo «deriveReq» e é originado da relação do tipo “abstração” da UML.

Relacionamento de refinamento: especifica que um elemento do modelo descreve as propriedades de um requisito em mais detalhes (WEILKIENS, 2007). É representado pelo estereótipo «refine» e é originado da relação do tipo “abstração” da UML.

Relacionamento de satisfação: descreve que um elemento de *design* satisfaz um requisito (WEILKIENS, 2007). Representado por um estereótipo «satisfy» e deriva da relação do tipo “realização” UML.

Relacionamento de rastreamento: é um relacionamento entre um requisito e um elemento arbitrário do modelo, que descreve uma relação geral (WEILKIENS, 2007). Esse relacionamento é utilizado apenas por razões de rastreabilidade. É representado pelo estereótipo «trace» e é originado da relação do tipo “abstração” da UML.

Relacionamento de Derivação: descreve que um elemento foi derivado de outro. É representado pelo estereótipo «derive» e é originado da relação do tipo “abstração” da UML.

Relacionamento de Realização: descreve que um elemento realiza o comportamento que o outro especifica. É representado por uma seta de linha

pontilhada e pode receber o estereótipo «*realize*». Este relacionamento também é originado da relação do tipo “abstração” da UML.

A Figura 37 ilustra os relacionamentos que podem existir entre os elementos de cada domínio por meio de um exemplo de um problema de um usuário de sapatos. Neste exemplo, o relacionamento tipo «*trace*» é utilizado para ligar Problemas, Necessidades e Requisitos e existe simplesmente por razões de rastreamento. A relação «*refine*», entre Requisitos e Casos de Uso indica que o Caso de uso define as propriedades do Requisito de maneira detalhada. Finalmente, o relacionamento entre Casos de Uso e Colaborações é do tipo «*realize*», que indica que a Colaboração realiza o Caso de Uso no domínio físico.

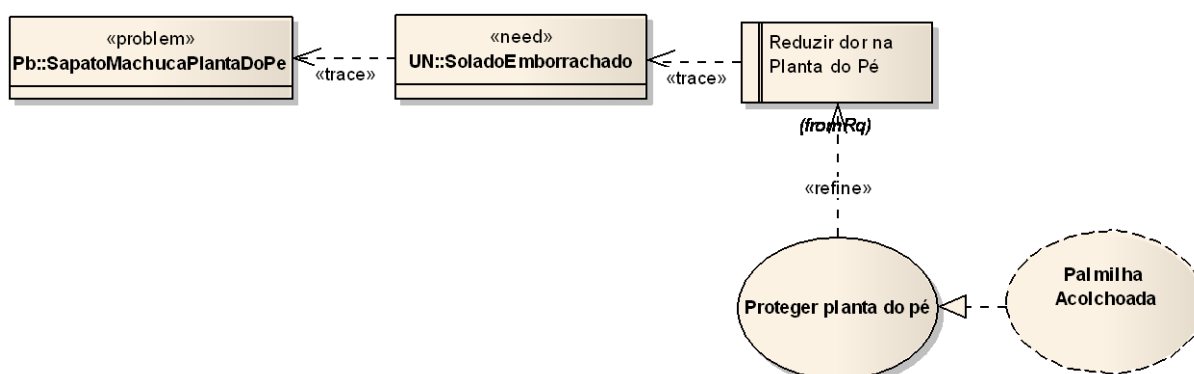


Figura 37 - Relacionamento Entre Elementos
Fonte: Autoria Própria

De acordo com o usuário deste exemplo, um dos seus problemas com sapatos é a “dor na planta do pé” e, na sua visão, a forma de solucionar o problema é adquirir sapatos novos. Assim, ele define uma necessidade que ele descreve como ter “solado emborrachado”. O analista de requisitos, por sua vez, buscou entender o problema do cliente e identificar uma necessidade que solucione este problema de maneira mais adequada. Assim, o ele define o requisito como “Reduzir a dor na planta do pé”. A partir disso foi definido um caso de uso “Proteger a planta do pé”. Como o requisito foi definido com foco em entender o verdadeiro problema do cliente, o analista de produto pode definir um parâmetro de projeto que é “Palmilha Acolchoada”. Este parâmetro resolve perfeitamente o problema e ainda permite que o sapato continue usando o solado antigo que tem maior qualidade. O objetivo deste exemplo é demonstrar que, embora as necessidades definidas pelo

cliente sejam relevantes, é importante tratá-las com cuidado para não limitar a solução. Desta forma, os analistas podem elaborar projetos com foco no problema correto e não nas impressões do cliente a respeito de uma possível solução.

Esta seção apresentou os principais conceitos e os elementos de modelagem UML/SysML que podem ser utilizados na aplicação da abordagem proposta. Uma vez que estes conceitos tenham sido identificados, podem ser definidas as etapas e atividades que compõem esta abordagem de especificação de requisitos.

4.6 Etapas do Modelo

A abordagem proposta sugere a decomposição de problemas e necessidades do cliente, seguindo o modelo do processo de decomposição de requisitos funcionais da Teoria de Projeto Axiomático (TPA). A seção anterior apresentou os níveis de abstração para o processo de decomposição dos elementos dos domínios de projeto. As quatro etapas desta abordagem foram definidas com base nestes conceitos. São elas: (1) Foco no Problema; (2) Foco nas Necessidades; (3) Foco nos Requisitos Essenciais; e (4) Foco nos Requisitos Técnicos. As etapas da abordagem proposta, suas entradas e saídas são ilustradas, por meio de um diagrama SDT, na Figura 38.

A primeira etapa tem o foco na análise e detalhamento no problema, além do entendimento do negócio do cliente. Esta etapa envolve a identificação de problemas, necessidades e requisitos de alto nível. Para cada problema são identificadas as necessidades e requisitos correspondentes. As entradas principais desta etapa são a documentação de negócio obtida do cliente e a lista de problemas e necessidades declaradas pelo cliente nos primeiros contatos com a equipe de desenvolvimento. Uma entrada secundária pode ser uma inconsistência nas matrizes de projetos da etapa seguinte, que obriga o analista a refazer a matriz. As saídas principais são a identificação dos problemas, necessidades e requisitos de alto nível. O controle desta etapa é o Axioma 1, que indica se os requisitos obtidos são bons e se estão de acordo com os problemas do cliente.

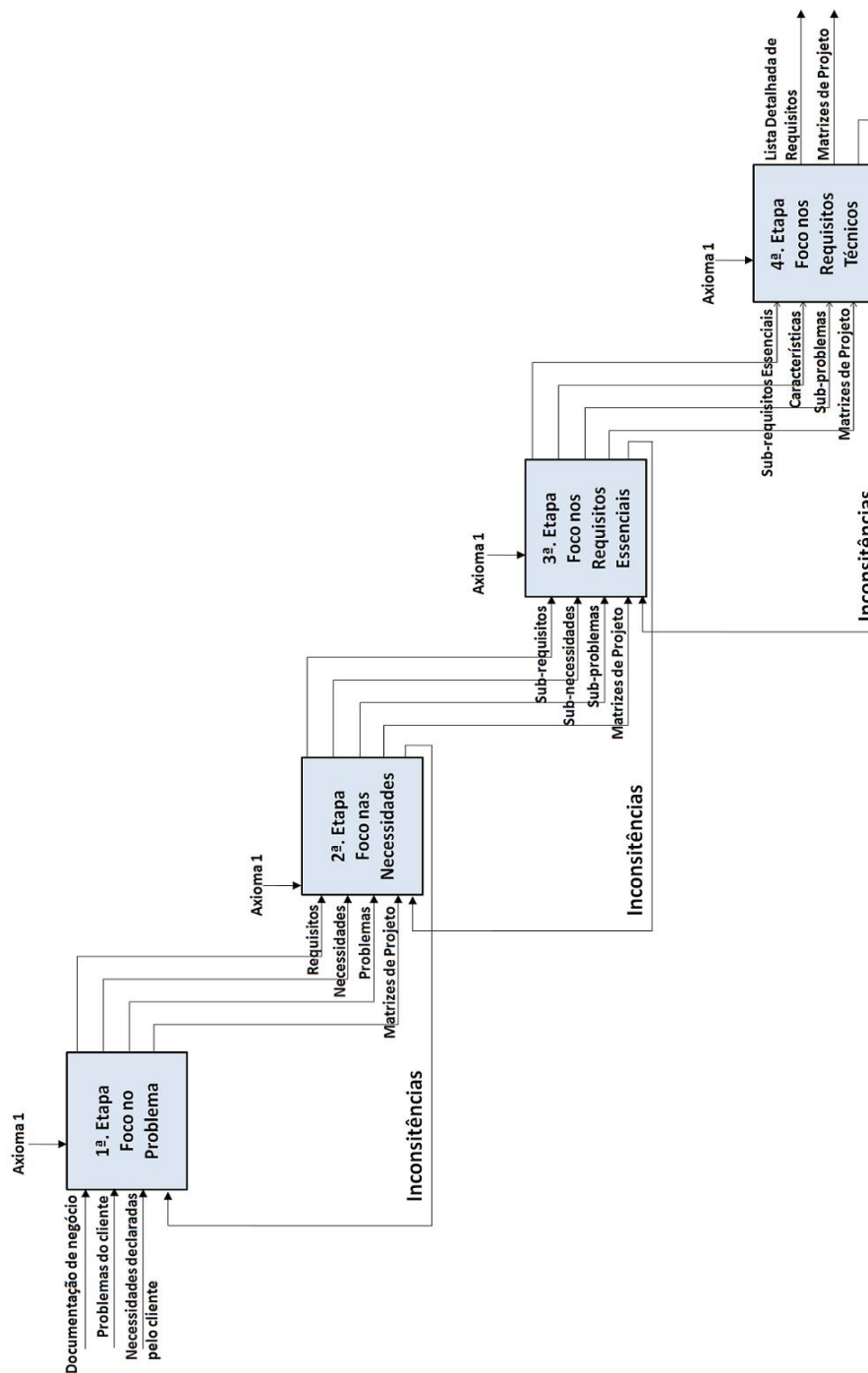


Figura 38 - Diagrama SDT das Etapas da abordagem
Fonte: Autoria Própria

A segunda etapa tem o foco na análise das necessidades do cliente, vez que os problemas já são conhecidos pelos analistas. Esta etapa envolve a identificação de subproblemas, subnecessidades e sub-requisitos por meio da decomposição das saídas da etapa anterior. Para cada subproblema são identificadas as subnecessidades e sub-requisitos correspondentes. As entradas principais desta etapa são a lista de problemas, necessidades e requisitos, bem como as matrizes de

projeto da etapa anterior. Uma entrada secundária pode ser uma inconsistência nas matrizes de projetos da etapa seguinte, que obriga o analista a refazer a matriz. As principais saídas são a identificação dos subproblemas, subnecessidades e sub-requisitos. O controle desta etapa também é o Axioma 1, que indica se os sub-requisitos obtidos são bons e se estão consistentes em relação às necessidades do cliente.

A terceira etapa tem o foco na análise e identificação dos requisitos essenciais do sistema. Esta etapa envolve a identificação de subproblemas, características e sub-requisitos essenciais por meio da decomposição das saídas da etapa anterior. Para cada subproblema são identificadas as características e sub-requisitos essenciais correspondentes. As entradas principais desta etapa são a lista de subproblemas, subnecessidades e sub-requisitos, bem como as matrizes de projeto da etapa anterior. Uma entrada secundária pode ser uma inconsistência nas matrizes de projetos da etapa seguinte, que obriga o analista a refazer a matriz. As principais saídas são a identificação dos subproblemas, características e sub-requisitos essenciais. O controle desta etapa é o Axioma 1, que indica se os sub-requisitos essenciais obtidos são bons e se são independentes entre si.

A quarta etapa tem o Foco na análise e identificação dos Requisitos Técnicos do sistema. Esta etapa envolve a identificação de subproblemas, características detalhadas e sub-requisitos técnicos por meio decomposição das saídas da etapa anterior. Para cada subproblema são identificadas as características detalhadas e sub-requisitos técnicos correspondentes. As entradas principais desta etapa são a lista de subproblemas, características e sub-requisitos essenciais, bem como as matrizes de projeto da etapa anterior. As principais saídas são a identificação dos subproblemas, características detalhadas e sub-requisitos técnicos. O controle desta etapa é o Axioma 1, que indica se os sub-requisitos técnicos obtidos são bons e se são consistentes e independentes entre si.

Esta seção descreveu as etapas da abordagem proposta de acordo com os níveis de detalhamento propostos para os elementos de cada domínio de projeto. A próxima seção apresenta as atividades realizadas neste processo de detalhamento e avaliação dos problemas, necessidades e requisitos com objetivo de elaborar um conjunto de requisitos de alta qualidade.

4.7 Atividades do Modelo Proposto

Conforme discutido anteriormente, os resultados obtidos no desenvolvimento de sistemas frequentemente causam desapontamentos aos clientes. Sabe-se que uma das maiores causas desses problemas está na dificuldade dos desenvolvedores em entender os objetivos para os quais o sistema será construído. De maneira geral, se um sistema não satisfaz seu usuário é porque foi projetado sem um correto entendimento de seu propósito.

A solução para esse tipo de problema costuma ser a realização de uma análise cuidadosa do objetivo da construção do sistema, antes que se inicie seu desenvolvimento. O método desenvolvido neste trabalho oferece um meio de se compreender melhor os problemas que o cliente pretende resolver por meio do sistema. Uma vez compreendidos os problemas, devem ser identificadas suas necessidades e, como consequência, especificar os requisitos adequados para o sistema. Na Figura 39 são apresentados os passos seguidos no modelo proposto de engenharia de requisitos, por meio de um diagrama de atividades.

1 - Analisar o Negócio do Cliente

Nesta atividade é estudado o negócio do cliente, com foco na área que utilizará o *software*. Isso significa entender o domínio do problema em que o sistema será inserido. Esta atividade é importante para colocar os desenvolvedores em sintonia com o negócio do cliente, de forma que, ao iniciar a análise referente ao problema a ser solucionado, possuam certa familiaridade com os produtos da empresa e com os termos técnicos relacionados ao negócio. Esta análise de negócio pode ser realizada por meio de visitas para conhecer os processos realizados, estudo de documentos internos, etc.

2 - Obter percepção do Cliente a Respeito dos Problemas e Necessidades

Nesta atividade são obtidas as percepções do cliente (normalmente são diversos *stakeholders*) a respeito dos problemas que precisam ser resolvidos e o que ele necessita ou espera do sistema que irá solucionar estes problemas. Isso significa, principalmente, obter informações a respeito dos problemas a serem

resolvidos. Esta atividade pode ser desenvolvida em uma ou mais reuniões com o cliente.

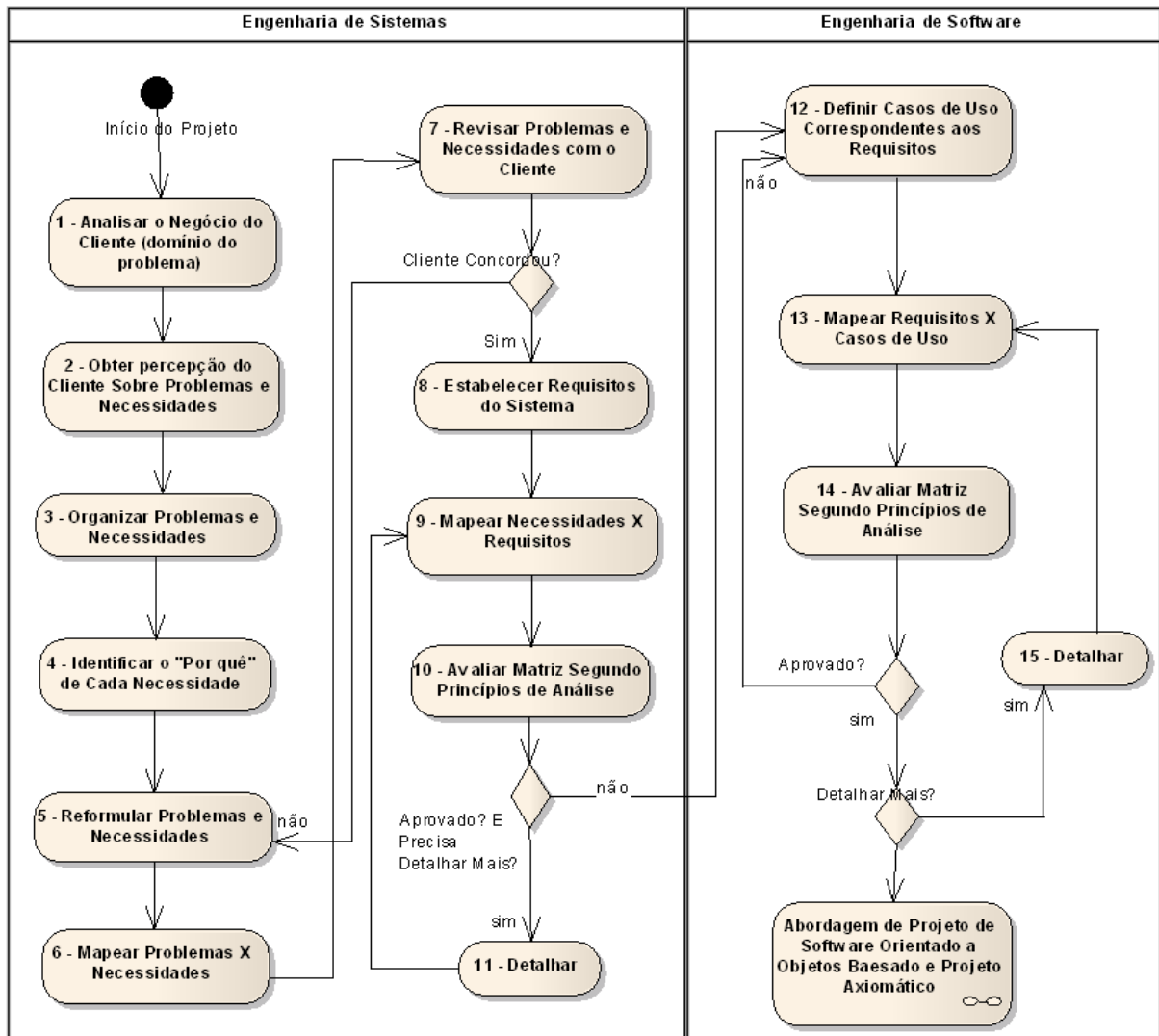


Figura 39 – Diagrama de Atividades Modelo Proposto
Fonte: Autoria Própria

3 - Organizar Problemas e Necessidades

De forma geral, as informações obtidas do cliente nestas primeiras reuniões costumam ser um tanto quanto confusas e desorganizadas. Por esta razão, o objetivo desta atividade é fazer com que a equipe de desenvolvimento organize estas informações colocando em listas separadas os problemas, as necessidades, os requisitos e até regras de negócio que são apresentadas pelo cliente. Organizar esta lista significa especialmente organizar a informação de forma que a equipe comece a vislumbrar com mais clareza a solução que deverá ser desenvolvida. Esta

atividade pode ser desenvolvida em uma reunião de equipe, desta forma, o fato de se discutir o conteúdo das diferentes listas irá ajudar a equipe a fortalecer seu entendimento do problema e da solução.

4 - Identificar o "Por que" de Cada Necessidade

Nas primeiras reuniões é bastante comum que o cliente apresente diversas necessidades e que não deixe claro que problemas cada necessidade irá solucionar. Por esta razão, nesta atividade deve ser realizada uma entrevista com o cliente com objetivo de interrogá-lo sobre a razão de cada uma destas necessidades existirem. Ao final da conversa espera-se ter identificado um problema para cada necessidade. Neste caso deverá ser estudada a relevância de se solucionar o problema.

5 - Reformular Problemas e Necessidades

Eventualmente, ao final da atividade número 4, é possível que ainda existam problemas e necessidades que não possuam um correspondente. Neste caso deverá ser avaliado se estes elementos devem ser eliminados ou se é necessário encontrar seu correspondente. Além disso, as respostas obtidas na atividade anterior deverão fornecer subsídios à equipe de desenvolvimento para reescrever os problemas e necessidades com base no entendimento obtido por meio da conversa com o cliente.

6 - Mapear Problemas X Necessidades

Nesta atividade os problemas e necessidades, devidamente reformulados, são colocados na matriz de projeto e seus relacionamentos são mapeados. Por padrão, busca-se manter os relacionamentos principais na linha diagonal da matriz e os relacionamentos secundários se distribuem na metade inferior e/ou superior. A prática de buscar manter todos os elementos não diagonais na parte inferior facilita a identificação dos elementos prioritários, especialmente no desenvolvimento. De maneira geral, os elementos mais à esquerda precisam ser realizados primeiro. Esta atividade pode ser realizada por apenas um dos analistas envolvidos no projeto por ser uma atividade bastante simples.

7 - Revisar Problemas e Necessidades com o Cliente

Nesta atividade a matriz de projeto elaborada na atividade anterior deve ser revisada pela equipe de projeto em conjunto com o cliente. Esta atividade é importante para assegurar que está claro como cada necessidade afeta a resolução de um ou mais problemas. A final desta atividade deve também ser avaliado se a matriz de projeto obtida está em acordo com o Axioma da Independência e com os princípios de análise estabelecidos na seção 4.9.

8 - Estabelecer Requisitos do Sistema

Esta atividade consiste em estabelecer os requisitos que atenderão as necessidades definidas até o momento. Recomenda-se que os requisitos sejam definidos por um analista, mas que sejam revisados por outras pessoas da equipe para assegurar que estejam em acordo com o conhecimento adquirido até este ponto do projeto.

9 - Mapear Necessidades X Requisitos

Nesta atividade as necessidades e os requisitos são colocados na matriz de projeto e seus relacionamentos são mapeados. Por padrão, busca-se manter os relacionamentos principais na linha diagonal da matriz e os relacionamentos secundários de distribuem na metade inferior e/ou superior. A prática de buscar manter todos os elementos não diagonais na parte inferior facilita a identificação dos elementos prioritários, especialmente no desenvolvimento. De maneira geral, os elementos mais à esquerda precisam ser realizados primeiro (LEE & JEZIOREK, 2006). Esta atividade pode ser realizada por apenas um dos analistas envolvidos no projeto, mas da mesma forma recomenda-se que seja revisada por uma ou mais pessoas da equipe.

10 - Avaliar Matriz Segundo Princípios de Análise

Esta atividade diz respeito à avaliação da matriz de projeto gerada na atividade anterior segundo o Axioma da Independência e os princípios de análise estabelecidos na seção 4.9. Nesta mesma atividade recomenda-se avaliar a matriz segundo alguns dos princípios de análise destacados na seção 6.6, que trata da discussão dos experimentos realizados.

11 – Detalhar

Nesta atividade, caso a matriz de projeto tenha sido aprovada segundo os princípios de análise e, caso seja considerado necessário, as necessidades e os requisitos são detalhados e, desta atividade, retorna-se à atividade número 9. Caso contrário segue-se para a próxima atividade.

12 - Definir Casos de Uso Correspondentes aos Requisitos

Esta atividade consiste em definir os casos de uso que refinarão os requisitos de sistema definidos até o momento. Recomenda-se que os casos de uso sejam definidos por um analista, mas que sejam revisados por outras pessoas da equipe para assegurar que sejam adequados para o projeto.

13 - Mapear Requisitos X Casos de Uso

Nesta atividade os requisitos e os casos de uso são colocados na matriz de projeto e seus relacionamentos são mapeados. Esta atividade pode ser realizada por apenas um dos analistas envolvidos no projeto, mas da mesma forma recomenda-se que seja revisada por uma ou mais pessoas da equipe.

14 - Avaliar Matriz Segundo Princípios de Análise

Esta atividade diz respeito à avaliação da matriz de projeto gerada na atividade anterior segundo o Axioma da Independência e os princípios de análise estabelecidos na seção 4.9. Nesta mesma atividade recomenda-se avaliar a matriz segundo alguns dos princípios de análise destacados na seção 6.6, que trata da discussão dos experimentos realizados.

15 - Detalhar

Nesta atividade, caso a matriz de projeto tenha sido aprovada segundo os princípios de análise e caso seja considerado necessário, as necessidades e os requisitos são detalhados. Desta atividade, retorna-se à atividade número 12. Caso contrário segue-se para a modelagem das colaborações ou outros elementos de modelagem definidos pela Abordagem de Projeto de *Software* Orientado a Objetos baseado em Projeto Axiomático (PIMENTEL, 2007). Por esta razão, as etapas seguintes, que envolvem o mapeamento de casos de uso para colaborações e modelagem do sistema, não são descritas neste trabalho.

4.8 Atividades de cada Etapa

É importante esclarecer que, de acordo com o foco de cada uma das etapas estabelecidas, algumas atividades podem ser mais relevantes que outras. Esta visão de foco das etapas e atividades está ligada à relação estabelecida entre o Projeto Axiomático e o Processo Unificado, discutidas anteriormente neste Capítulo. Conforme apresentado na seção correspondente ao tema cada fase tem mais ênfase no relacionamento entre dois domínios. Por esta razão as etapas desta abordagem estão mais focadas em determinadas atividades que outras.

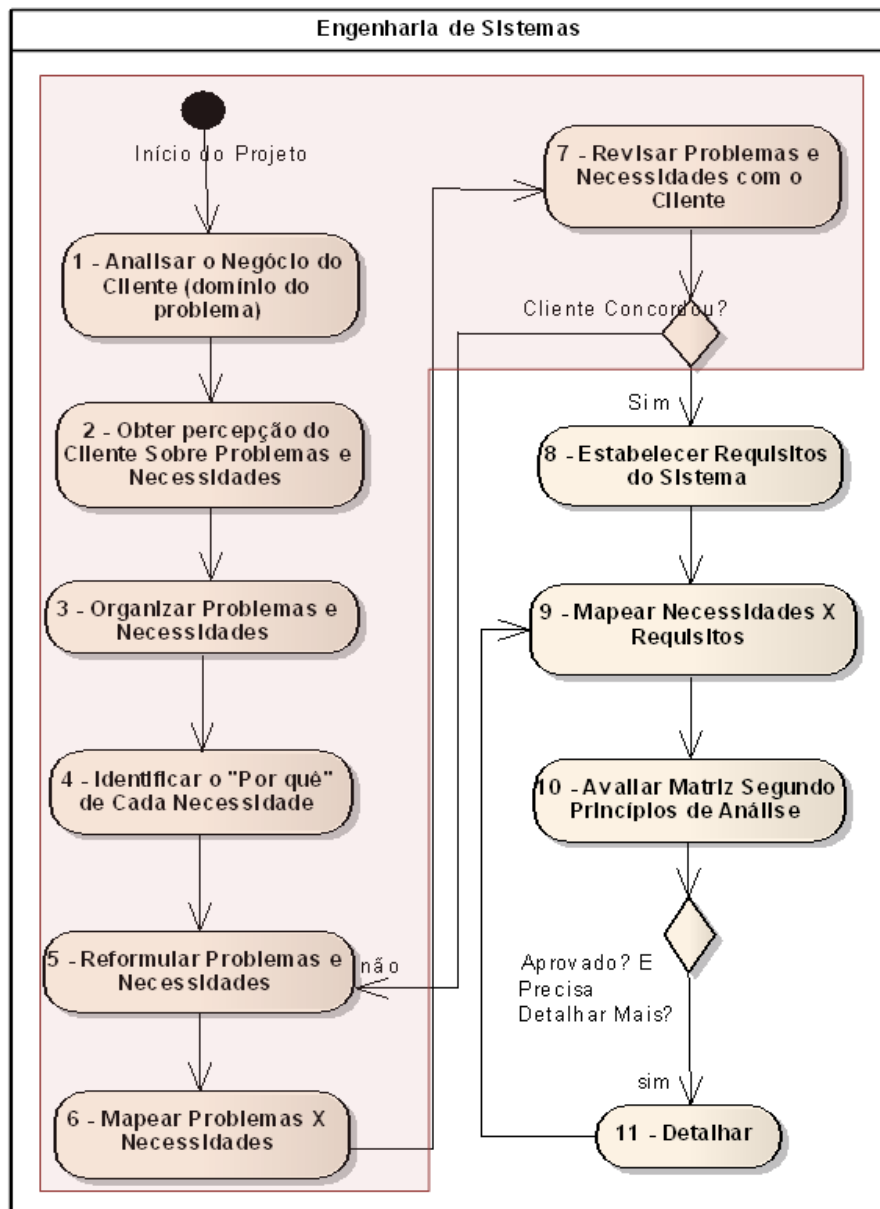


Figura 40 - Atividades com Foco no Problema
 Fonte: Autoria Própria

A primeira etapa tem seu foco no problema e por esta razão nesta fase as atividades de 1 a 7 são as mais relevantes, conforme ilustra a Figura 40. Esta etapa pode ser comparada à atividade de “Análise do Problema” do Fluxo de Requisitos do Processo Unificado, pois é especialmente nesta fase que os analistas se preocupam em entender quais são os problemas do cliente.

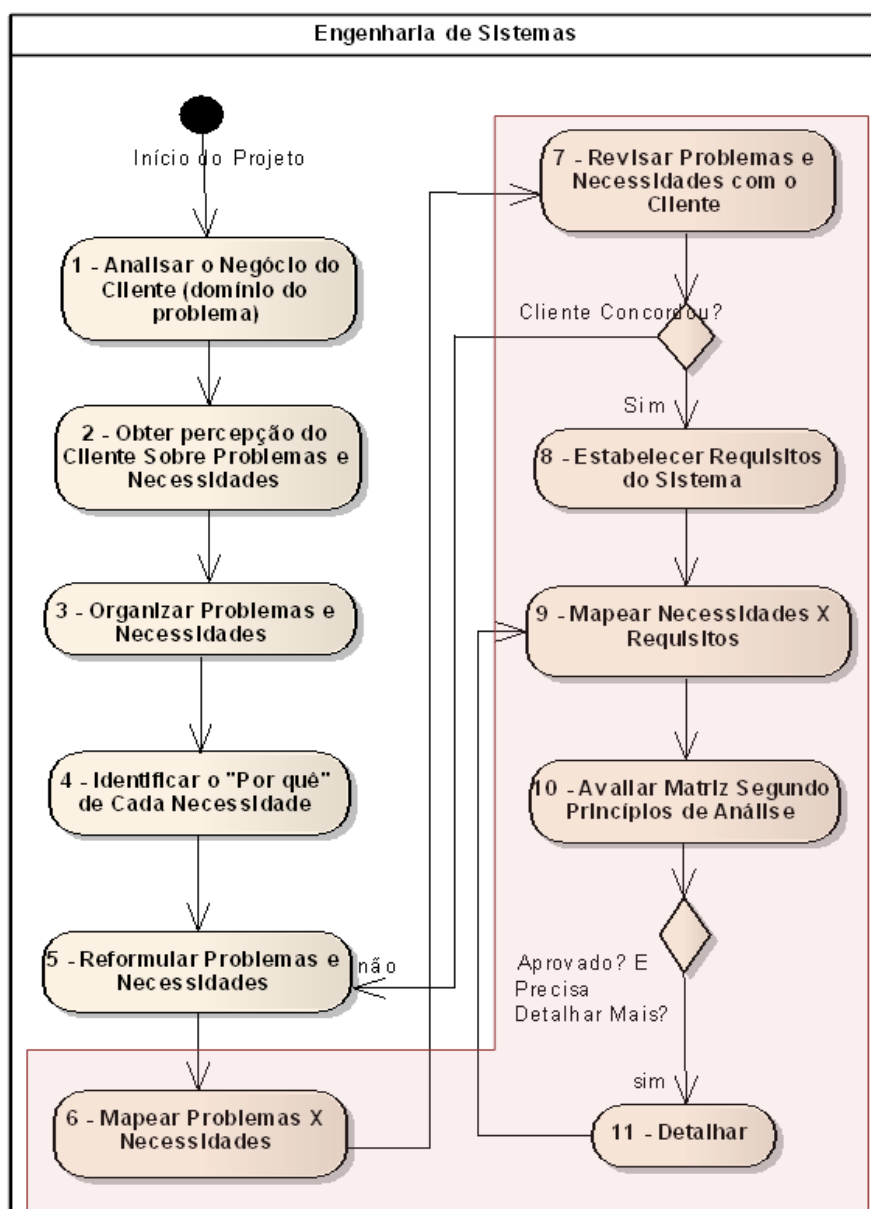


Figura 41 - Atividades com Foco nas Necessidades
Fonte: Autoria Própria

Por sua vez, a segunda etapa, que tem foco nas necessidades, dá mais relevância às atividades 6 a 11. Esta etapa pode ser comparada à atividade de “Compreender as Necessidades do Cliente” do Fluxo de Requisitos do Processo

Unificado. Isso se deve ao fato de que nesta fase que os analistas se preocupam principalmente em entender quais são as necessidades do cliente que serão atendidas por meio do sistema.

A terceira e quarta etapas têm foco nos requisitos e dão mais relevância às atividades 8 a 15. Estas etapas podem ser comparadas à atividade de “Definir o Sistema” do Fluxo de Requisitos do Processo Unificado. Nestas duas fases os analistas se preocupam principalmente em entender quais os requisitos do sistema. O que as diferencia é que na terceira os requisitos são independentes da solução técnica enquanto na quarta os requisitos são dependentes da solução.

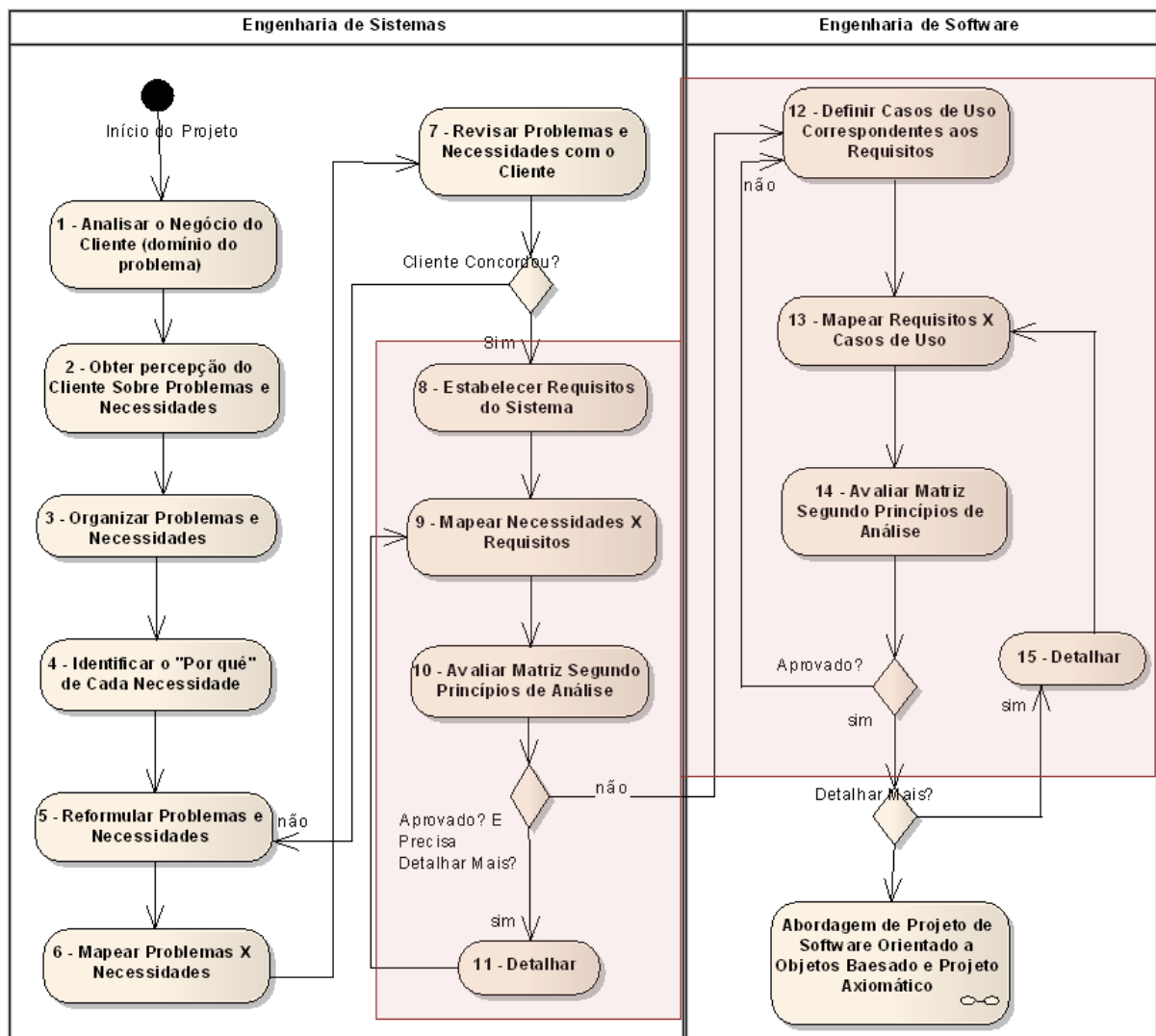


Figura 42 - Atividades com Foco nos Requisitos
 Fonte: Autoria Própria

Nesta seção apresentou-se as atividades desenvolvidas na abordagem de especificação de requisitos proposta nessa dissertação. Destacou-se que cada atividade pode ter mais ou menos relevância nas etapas de decomposição do projeto. A seguir são apresentados alguns princípios de análise de matrizes identificados por meio dos experimentos realizados.

4.9 Princípios de Análise

Esta seção apresenta um estudo dos relacionamentos que podem existir dentro de uma matriz de projeto e uma discussão a respeito do significado de cada uma e da forma como tratá-las. Estes relacionamentos foram identificados e estudados conforme os resultados dos experimentos estudados no próximo capítulo. Boa parte deles haviam sido discutidos anteriormente pois estão relacionados aos Teoremas de Projeto Axiomático (SUH, 1990). Estes princípios podem ser usados como um guia para a avaliação da qualidade da matriz de projeto.

Embora tenham sido identificados estes princípios de análise para as matrizes e seus respectivos significados, não foi possível realizar mais experimentos para comprovar a eficiência de se manter a independência nos domínios envolvidos nesta abordagem. Para resolver esta pendência, um estudo mais aprofundado a respeito deste assunto pode ser tema de um trabalho futuro.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
N1	X		X						
N2		X							
N3	X		X						
N4				X					
N5				X	X				
N6						X	X		
N7								X	
N8								X	
N9									

Figura 43- Tipos de Relacionamentos em uma Matriz de Projeto
Fonte: Autoria Própria

Na Figura 43 uma célula da matriz marcada com “X” indica que o Requisito daquela coluna atende a Necessidade correspondente à linha. É importante lembrar que para exemplificação foram utilizados os domínios de Cliente e Funcional, mas

as mesmas regras são válidas para o relacionamento entre todos os domínios. Outra observação fundamental é que, para efeito desta discussão, é necessário que nos relacionamentos analisados sempre existam elementos diagonais (LEE & JEZIOREK, 2006). Os elementos diagonais são aqueles que se encontram exatamente sobre a linha diagonal da matriz. Eles representam o relacionamento principal entre Requisitos e Necessidades, neste exemplo. Os demais relacionamentos podem ser denominados secundários e, de maneira geral, costumam representar uma ligação de dependência menor que o relacionamento principal.

Caso 1: Relacionamento cíclico ou acoplado

Quando os relacionamentos formam um ciclo, isso indica acoplamento entre Requisitos e Necessidades, como ilustram as necessidades N1 e N3 na Figura 43. Esse relacionamento pode indicar que há um erro de interpretação das necessidades, que poderiam ser representados por uma única necessidade; ou que há um erro de definição dos requisitos que devem atender estas necessidades. No caso acima estão envolvidos apenas dois elementos diagonais, mas pode-se ter vários itens formando um ciclo (LEE & JEZIOREK, 2006), como no exemplo da Figura 44. Sempre que um relacionamento deste tipo for encontrado deve ser eliminado, para que o projeto esteja de acordo com o Axioma da Independência.

	R1	R2	R3	R4	R5
N1	X				X
N2	X	X			
N3			X		
N4	X	X		X	
N5		X			X

Figura 44 - Relacionamento Acoplado
Fonte: Autoria Própria

Caso 2: Relacionamento triangular ou semiacoplado

A Figura 43 ilustra, com as necessidades N4 e N5, o caso em que os relacionamentos das células formam um triângulo indicando um semiacoplamento que é aceitável, vez que seus elementos não formam nenhum ciclo.

Caso 3: Relacionamento múltiplo em linha

Este relacionamento ocorre quando os relacionamentos das células formam uma linha com dois ou mais elementos marcados, sem que existam mais ligações em nenhuma das colunas envolvidas, conforme ilustra a necessidade N6 da Figura 43. Isso pode indicar que a necessidade poderia ser dividida em subnecessidades ou que os requisitos deveriam ser reinterpretados para formarem apenas um requisito. Sempre que um relacionamento deste tipo for encontrado deve ser eliminado, por estar ligado ao Teorema 3 (Anexo A).

Caso 4: Relacionamento múltiplo em coluna

Este caso, representado na Figura 43 pelo requisito R8, ocorre quando os relacionamentos das células formam uma coluna com dois ou mais elementos marcados com X, desde que não existam mais ligações em nenhuma das linhas envolvidas. Isso pode indicar que o requisito poderia ser dividido em sub-requisitos ou que as necessidades deveriam ser reinterpretadas para formarem apenas uma. Este tipo de situação deve ser evitado e costuma estar relacionada ao Teorema 1 (Anexo A).

Caso 5: Ausência de relacionamentos em uma coluna

A ausência de relações em uma coluna, ilustrado pelo requisito R9 da Figura 43, pode significar que o requisito não existe ou que a necessidade correspondente não foi identificada. Este tipo de situação costuma estar relacionada ao Teorema 3 (Anexo A) e deve ser eliminado.

Caso 6: Ausência de relacionamentos em uma linha

A necessidade N9 da Figura 43 representa o caso em que ocorre a ausência de relações em uma linha. Isso pode significar que a necessidade não existe ou que o requisito para atender a necessidade não foi definido. Este tipo de situação costuma estar relacionada ao Teorema 1 (Anexo A) e deve ser, preferencialmente, eliminado.

4.10 Protótipo da Ferramenta

Um dos objetivos desta dissertação foi desenvolver um protótipo de uma ferramenta para dar suporte à aplicação da abordagem proposta. Nesta versão a ferramenta possui a funcionalidade de cadastro de projetos e de seus componentes, como problemas, necessidades e requisitos. Além disso, permite avaliar a independência funcional das diferentes matrizes que podem ser obtidas ao se relacionar estes elementos. A Figura 45 apresenta a tela principal do protótipo desenvolvido.

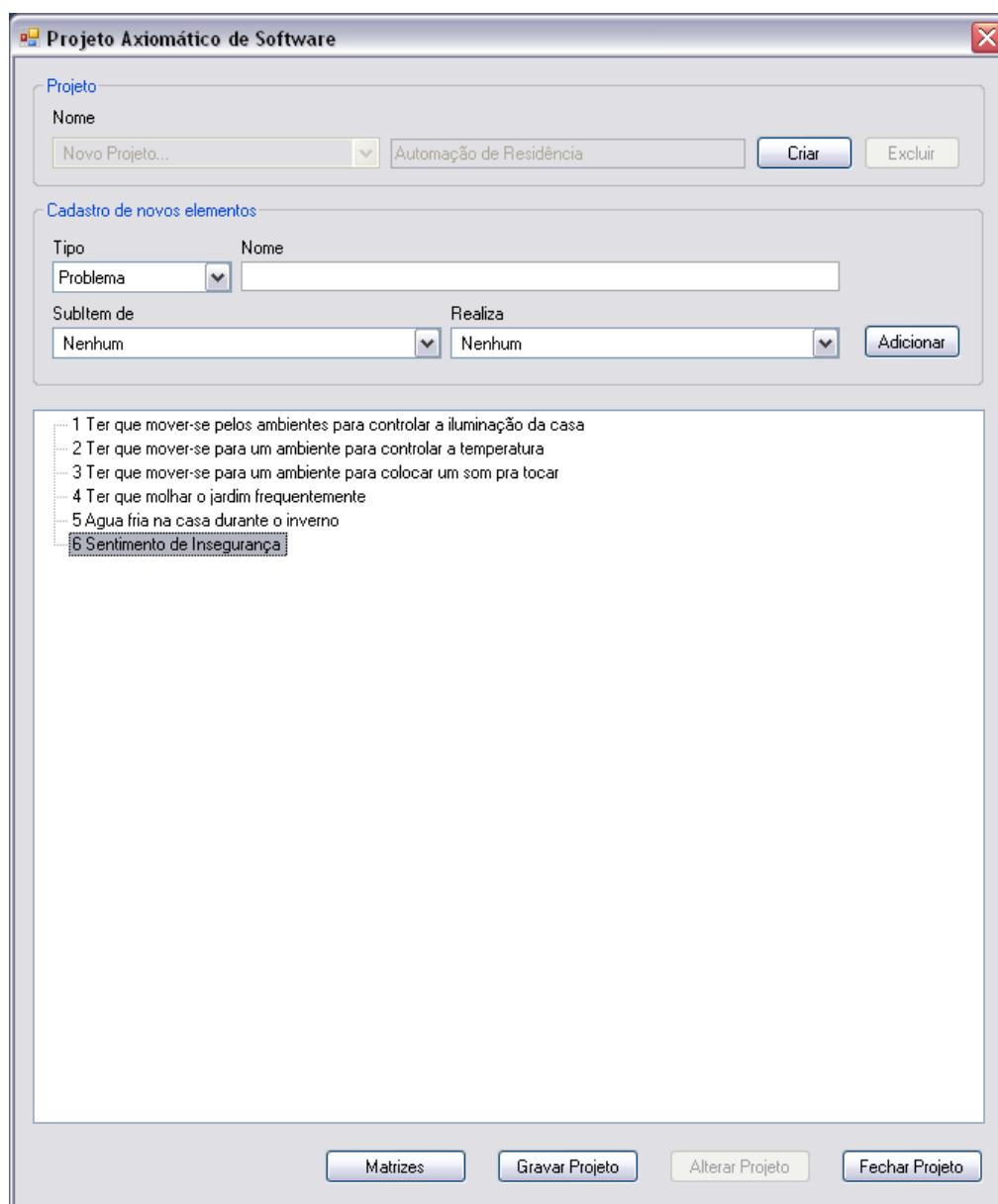


Figura 45 - Tela Principal
Fonte: Autoria Própria

O protótipo toma por base a reangularidade e a semangularidade para a avaliação das matrizes. Esses dois valores são calculados por meio das equações apresentadas em Suh (1990) (ver anexo A). Ambas as equações possuem um valor máximo de 1, que corresponde a um projeto desacoplado (ideal), assim, um valor 0 corresponde a um projeto acoplado (inaceitável). Quando a independência funcional diminui, o valor de “reangularidade” e de “semangularidade” decresce.

Nesta versão da ferramenta não foram incluídos os princípios de análise estabelecidos na seção anterior. Isso se deve ao fato que não foi possível estabelecer neste trabalho uma equação que levasse em conta estes princípios, ficando esta atividade para um trabalho futuro. O desenvolvimento de uma equação que avalie a matriz segundo os princípios de análise pode ser objeto de trabalhos futuros. No Apêndice A desta dissertação a ferramenta é apresentada em maiores detalhes.

4.11 Conclusão

Este capítulo apresentou a abordagem proposta de especificação de requisitos de sistemas baseada em Projeto Axiomático que pode ser aplicada em conjunto com a abordagem de projeto de *software* orientado a objetos baseada na teoria de Projeto Axiomático. A definição desta abordagem foi o objetivo principal desta dissertação de mestrado.

A aplicação dessas duas abordagens em conjunto com o Processo Unificado pode trazer vantagens ao desenvolvimento de *software* orientado a objetos. Estas vantagens podem ser, por exemplo, oferecer critérios para verificar inconsistências de requisitos, para a identificação de um conjunto de requisitos adequado e, também, para a escolha da melhor solução de projeto para atender estes requisitos. Além disso, podem ser descritas outras vantagens como fornecer mecanismos de rastreabilidade de componentes em relação aos requisitos funcionais e dos requisitos funcionais em relação aos problemas e necessidades que lhes deram origem.

Foram definidas correspondências entre conceitos do Projeto Axiomático e conceitos do Processo Unificado. Por meio desta análise, pode ser verificada a

correspondência entre o processo de mapeamento entre domínios do Projeto Axiomático e fases e fluxos de desenvolvimento do Processo Unificado. O que confirma a possibilidade de utilizar as duas metodologias em conjunto. A abordagem proposta neste trabalho se insere nas atividades de análise de negócio e especificação de requisitos que são realizadas, principalmente, nas fases de concepção e elaboração do projeto.

A abordagem proposta é realizada em quatro etapas de acordo com níveis de abstração definidos para os conceitos de Problema, Necessidade e Requisito. De acordo com estes níveis de abstração, a cada etapa são realizadas atividades de decomposição e aplicação do primeiro axioma. Cada etapa possui um foco diferente, a primeira tem foco no problema, a segunda nas necessidades, a terceira e a quarta nos requisitos. De acordo com o foco da etapa muda a ênfase nas atividades de mapeamento e detalhamento dos diferentes domínios.

Desta forma, obteve-se um processo completo de especificação de requisitos que envolve a análise domínio do problema em que está inserido o cliente, o entendimento das necessidades deste cliente e, por fim, a definição dos requisitos que o sistema deve atender para resolver os problemas do cliente.

Este capítulo apresentou também alguns princípios de análise de matrizes de projeto que devem ser observados cada vez que uma nova matriz for elaborada. Além disso, foi apresentado outro resultado do trabalho que foi um protótipo de uma ferramenta de análise de matrizes de projeto. A seguir é apresentado um caso de estudo que exemplifica a aplicação da abordagem proposta.

5 Caso de estudo

Este capítulo apresenta um caso de estudo para exemplificar e avaliar a aplicação da abordagem de especificação de requisitos proposta nesta dissertação. Neste caso de estudo é apresentado o projeto de um *software* que serve como base para avaliar se o uso da abordagem proposta permite aplicar os princípios da Teoria de Projeto Axiomático à especificação de requisitos. Neste capítulo são apresentados os objetivos do caso de estudo, a descrição do sistema, o passo a passo da elaboração do projeto e as conclusões obtidas.

5.1 Objetivos do Caso de estudo

O objetivo geral deste caso de estudo é exemplificar a aplicação da abordagem de especificação de requisitos a um projeto de sistema de *software*, passando pelas diversas etapas do projeto. Além disso, deseja-se avaliar a aplicação da abordagem proposta nesta dissertação do ponto de vista da praticidade de aplicação e efetividade dos resultados.

5.2 Descrição do Sistema

O sistema utilizado como exemplo neste caso de estudo corresponde a um sistema de teste de equipamentos em uma linha de produção. O objeto deste caso de estudo foi escolhido por ser um sistema que foi construído para solucionar diversos problemas do cliente e, por esta razão, envolver de forma clara todos os domínios envolvidos na abordagem proposta. O cliente possuía um sistema de teste em uso na linha, porém ele apresentava diversos problemas. Embora o caso de estudo seja sobre um caso verídico ele foi realizado após a implementação do sistema real, servindo somente para fins acadêmicos.

O propósito principal do sistema é testar os equipamentos produzidos pela empresa que são mini-impressoras térmicas ou matriciais. Além das impressoras montadas, o sistema deve testar as placas produzidas pelo fornecedor logo após a montagem das mesmas. Além disso, o cliente deseja resolver problemas que

encontra no sistema atual como: dificuldade de modificar baterias de testes por modelo de equipamento, interface difícil de compreender e falta de informação a respeito dos testes realizados.

O desenvolvimento deste caso de estudo foi realizado utilizando-se a ferramenta de modelagem *Enterprise Architect 7.0* em concomitância com o protótipo desenvolvido para a avaliação de matrizes de projeto. O uso deste protótipo se fez necessário por que não era possível avaliar as matrizes de projeto por meio do EA uma vez que esta ferramenta não é apropriada para aplicação da Teoria de Projeto Axiomático. Finalmente, a pessoa que desempenhou o papel de analista no desenvolvimento deste caso de estudo foi a autora deste trabalho.

5.3 Descrição do Caso de estudo

Este caso de estudo foi desenvolvido seguindo as etapas da abordagem proposta de especificação de requisitos. A pessoa responsável pela realização do estudo de caso foi o autor deste documento. Inicialmente foi estudado o negócio de que se tratava e os produtos do cliente. A seguir foram identificados os problemas e necessidades do cliente. A partir desta definição, foi iniciada a aplicação da abordagem passando pelos diversos níveis de decomposição. Seguindo a abordagem, a cada etapa foram identificados os elementos de cada domínio de acordo com a ênfase da fase. Também, seguindo a abordagem, a cada etapa foi realizada a aplicação do Axioma da Independência às matrizes obtidas. A identificação dos elementos foi realizada com uso da ferramenta de modelagem EA. Os dados foram posteriormente transferidos para o protótipo que avaliava as matrizes de projeto.

5.3.1 Etapa 1 – Foco no problema

O projeto se inicia com o primeiro nível de decomposição dos problemas, necessidades e requisitos, que corresponde à primeira etapa da abordagem proposta, do foco no problema. Na primeira atividade desta etapa é estudado o negócio do cliente com foco na área que utilizará o sistema. Neste caso, o cliente é uma empresa fabricante de equipamentos de impressão. Na análise do negócio

foram realizadas visitas à fábrica, estudados os processos de teste, estudado o sistema que estava em uso na empresa e, também, o equipamento que seria testado por meio do *software*. A seguir, em reuniões com o cliente, foi obtida a percepção do cliente em relação aos problemas que ele possuía e às características que necessitavam do *software*. Em seguida foram organizadas as listas de problemas e necessidades.

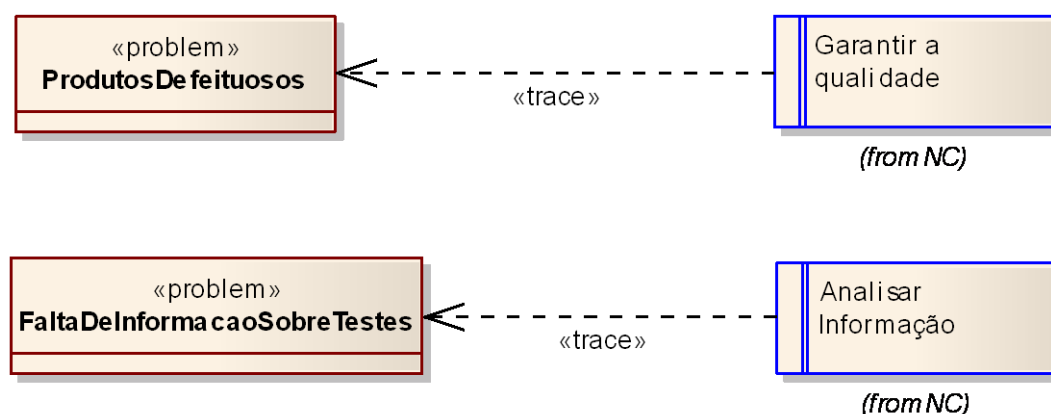


Figura 46 - Diagrama Problema X Necessidades, Etapa 1
Fonte: Autoria Própria

Como o cliente tinha bastante claro seus principais problemas, não foi necessário realizar a atividade 4 do modelo proposto, que é perguntar o “por que” de cada necessidade. Assim, modelou-se os problemas e necessidades na ferramenta *Enterprise Architect* (EA), resultando no primeiro diagrama do projeto que relaciona os problemas e necessidades por meio de um relacionamento do tipo «*trace*». A Figura 46 apresenta uma parte do diagrama obtido nesta primeira etapa.

Matriz de Projeto					
Tipo de matriz	Problemas X Necessidades		Nível	1	
	Reangularidade: 1			Semangularidade: 1	
Problema \ Necessidade	1 Sistema com um mecanismo de atualização automática de versões, como os de anti-vírus.	2 Necessita-se que os testes sejam configuráveis	3 Analisar informação dos testes para identificar as principais causas de erros.	4 Necessita-se que o sistema facilite o entendimento do usuário	5 Garantir a Qualidade dos produtos por meio dos testes.
1 Atualização de SW frequentemente gera problemas.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2 Depende-se dos desenvolvedores para qualquer alteração ...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3 A empresa desconhece os principais problemas que ocorre...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4 O sistema atual é difícil de entender e causa confusões.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5 Defeitos são comuns na produção, mas se a empresa vend...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figura 47 - Matriz de projeto, Etapa 1
Fonte: Autoria Própria

Após a elaboração do diagrama, que realizou o mapeamento dos problemas e necessidades, elaborou-se a matriz com o apoio do protótipo desenvolvido. A matriz obtida possui reangularidade igual a 1 e semangularidade também igual a 1, o que a torna uma matriz desacoplada. A matriz é ilustrada na Figura 47.

Como pode ser observado na Figura 47, em um primeiro momento foram identificados cinco problemas e cinco necessidades, que deveriam solucionar estes problemas. Os cinco problemas identificados foram: “1 Atualização de SW frequentemente gera problemas”, “2 Depende-se dos desenvolvedores para qualquer alteração do sistema”, “3 A empresa desconhece os principais problemas que ocorrem na produção”, “4 O sistema atual é difícil de entender e causa confusões” e “5 Defeitos são comuns na produção, mas se a empresa vender produtos com defeitos perderá espaço no mercado”. As cinco necessidades identificadas foram: “1 Sistema com um mecanismo de atualização automática de versões, como os de antivírus”, “2 Necessita-se que os testes sejam configuráveis”, “3 Analisar informação dos testes para identificar as principais causas de erros”, “4 Necessita-se que o sistema facilite o entendimento do usuário” e “5 Garantir a Qualidade dos produtos por meio dos testes”.

Além dos problemas e necessidades, nesta etapa também foram identificados os requisitos de primeiro nível: “1 Atualização automática do *software*”, “2 Configuração de testes”, “3 Modulo de Relatórios de Testes”, “4 Interface Visual e Sonora Compreensível” e “5 Teste de Equipamentos”. Estes requisitos foram evoluindo em conjunto com os problemas e necessidades, até a fase 3 em que o foco passa a estar nos requisitos como é visto mais adiante neste capítulo.

A atividade seguinte desta etapa foi detalhar os problemas identificados, o resultado deste detalhamento é parcialmente ilustrado pela Figura 48. Assim, de acordo com a figura, o problema que dizia que “4 O sistema atual é difícil de entender e causa confusões.” foi detalhado de maneira que seu significado ficasse mais claro. Desta forma, o problema foi detalhado em dois subproblemas “4.1 Menu do sistema atual são confusos” e “4.2 Difícil entender se o resultado dos testes foi aprovado ou reprovado”.

Da mesma forma, o problema “3 A empresa desconhece os principais problemas que ocorrem na produção” foi decomposto em dois subproblemas: “3.1 Falta Informação sobre os principais Defeito que ocorrem nas Placas” e “3.2 Falta Informação sobre os principais Defeito que ocorrem com as Impressoras”. O

problema “5 Defeitos são comuns na produção” foi decomposto em dois subproblemas: “5.1 Placas chegam dos fornecedores com Defeitos” e “5.2 Impressoras saem com defeitos da produção”.

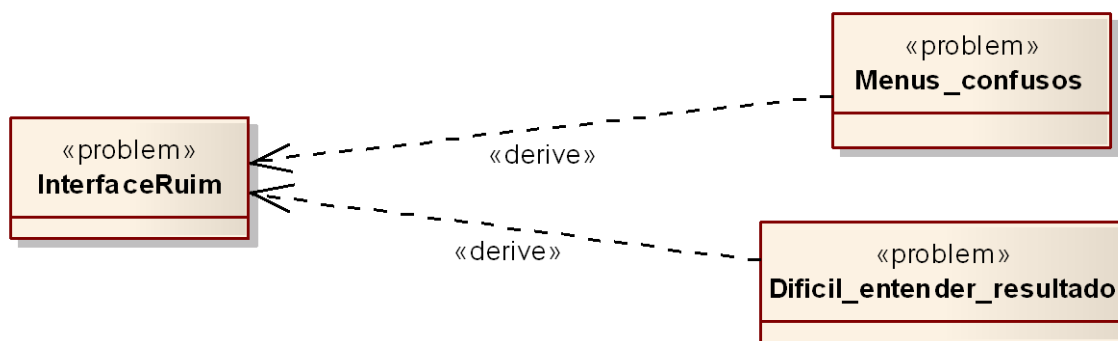


Figura 48 - Detalhamento dos problemas, Etapa 1
Fonte: Autoria Própria

Uma vez detalhados os problemas da primeira etapa, passou-se a detalhar as necessidades. Esta atividade de detalhamento marca a passagem para a Etapa 2, do Foco nas Necessidades.

5.3.2 Etapa 2 – Foco nas Necessidades

Na segunda etapa da abordagem, a primeira atividade realizada foi a decomposição das Necessidades do Cliente. Nesta atividade foram identificadas algumas decomposições das necessidades, como ilustra a Figura 49. Assim, a necessidade “4 Necessita-se que o sistema facilite o entendimento do usuário” foi detalhada em duas subnecessidades: “4.1 Identificar Modelo do Equipamento por Numero Serial” e “4.2 Emitir Sons Indicativos de Resultados negativos de testes”.

Da mesma maneira, a necessidade “3 Analisar informação dos testes para identificar as principais causas de erros” foi detalhada em duas necessidades “3.1 Analisar os resultados de Testes de Placas” e “3.2 Analisar os resultados de Testes de Impressoras”. A necessidade “5 Garantir a Qualidade dos produtos por meio dos testes” foi detalhada em duas necessidades “5.1 Garantir que as placas saiam do fornecedor sem defeitos” e “5.2 Garantir que as impressoras saiam da fabrica sem defeitos”.

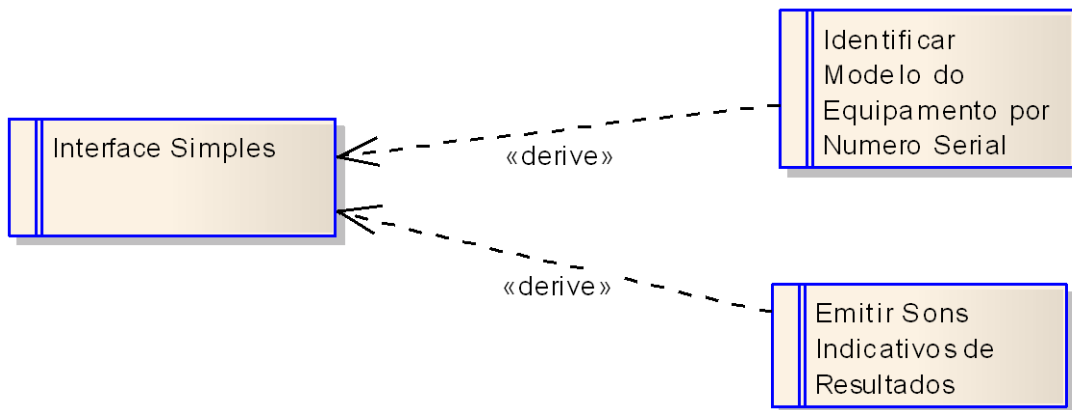


Figura 49 - Detalhamento das Necessidades, Etapa 2
Fonte: Autoria Própria

A atividade seguinte desta etapa foi mapear os relacionamentos entre os problemas e necessidades por meio de um relacionamento do tipo «trace» na ferramenta EA. O diagrama obtido é ilustrado parcialmente na

Figura 50.

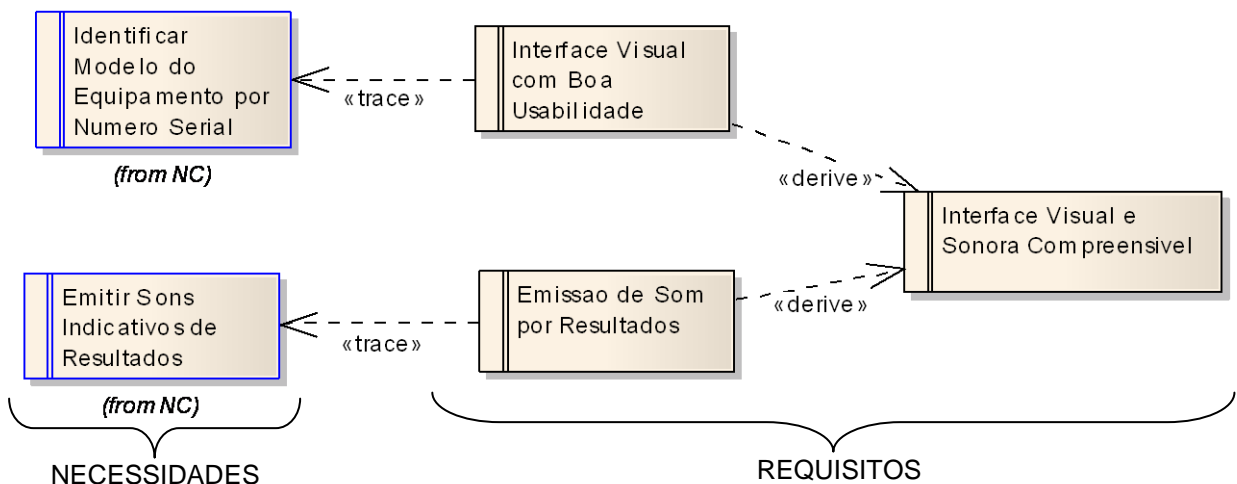


Figura 50 – Necessidades X Requisitos, Etapa 2
Fonte: Autoria Própria

Em seguida foi elaborada a matriz de projeto com o protótipo desenvolvido. A matriz obtida possui reangularidade igual a 1 e semangularidade também igual a 1, o que a torna uma matriz desacoplada como ilustra a Figura 51.

Acompanhando os detalhamentos realizados na segunda etapa, os requisitos também foram detalhados. Os sub-requisitos obtidos foram os seguintes: “3.1 Relatório Teste de Placas”, “3.2 Relatório Teste de Impressoras”, “4.1 Identificar Modelo do Equipamento por Número Serial”, “4.2 Emissão de Som para falha em

Teste”, “5.1 Teste de Placas” e “5.2 Teste de Impressoras”. Nesta etapa também começaram a ser definidos casos de uso correspondentes aos requisitos.

Matriz de Projeto									
Tipo de matriz		Problemas X Necessidades		Nível		2		Reangularidade: 1	Semangularidade: 1
Problema \ Necessidade	1 Sistema com um mecanismo de atualização automática de versões, como os de anti-vírus.	2 Necessita-se que os testes sejam configuráveis	3.1 Analisar os resultados de Testes de Placas	3.2 Analisar os resultados de Testes de Impressoras	4.1 Identificar Modelo do Equipamento por Numero Serial	4.2 Emitir Sons Indicativos de Resultados negativos	5.1 Garantir que as placas saiam do fornecedor sem defeitos	5.2 Garantir que as impressoras saiam da fabrica sem defeitos	
1 Atualização de SW frequentemente gera proble...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2 Depende-se dos desenvolvedores para qualque...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3.1 Falta Informacao sobre os principais Defeito q...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3.2 Falta Informacao sobre os principais Defeito q...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4.1 Menus do sistema atual são confusos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4.2 Difícil entender se o resultado dos testes foi ap...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.1 Placas chegam dos fornecedores com Defeitos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5.2 Impressoras saem com defeitos da produção	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Figura 51 - Matriz de projeto, Etapa 2

Fonte: Aatoria Própria

Uma vez aprovada a matriz de projeto desta etapa, passou-se a detalhar as necessidades novamente. Esta atividade de detalhamento marcou a passagem para a Etapa 3, do Foco nos Requisitos Essenciais.

5.3.3 Etapa 3 – Foco nos Requisitos Essenciais

Na terceira etapa da abordagem, a primeira atividade foi decompor novamente as necessidades do cliente. Nesta atividade foram identificadas algumas decomposições das subnecessidades da etapa anterior. Assim, a necessidade “5.1 Garantir que as placas saiam do fornecedor sem defeitos” foi decomposta nas seguintes características: “5.1.1 Garantir que a Memória está Funcionando” e “5.1.2 Garantir que a Interface Serial Funciona”. Por sua vez, a subnecessidade “5.2 Garantir que as impressoras saiam da fabrica sem defeitos” foi detalhada nas seguintes características: “5.2.1 Garantir que o Sensor de Papel Funciona”, “5.2.2 Garantir que a o Sensor de Tampa Funciona”, “5.2.3 Garantir que a Guilhotina Funciona”, “5.2.4 Garantir que a Impressão de Imagens Funciona” e “5.2.5 Garantir que a Impressão de Caracteres Funciona”. A modelagem destas características e

seu relacionamento com os elementos dos níveis anteriores na ferramenta EA é parcialmente ilustrada na Figura 52.

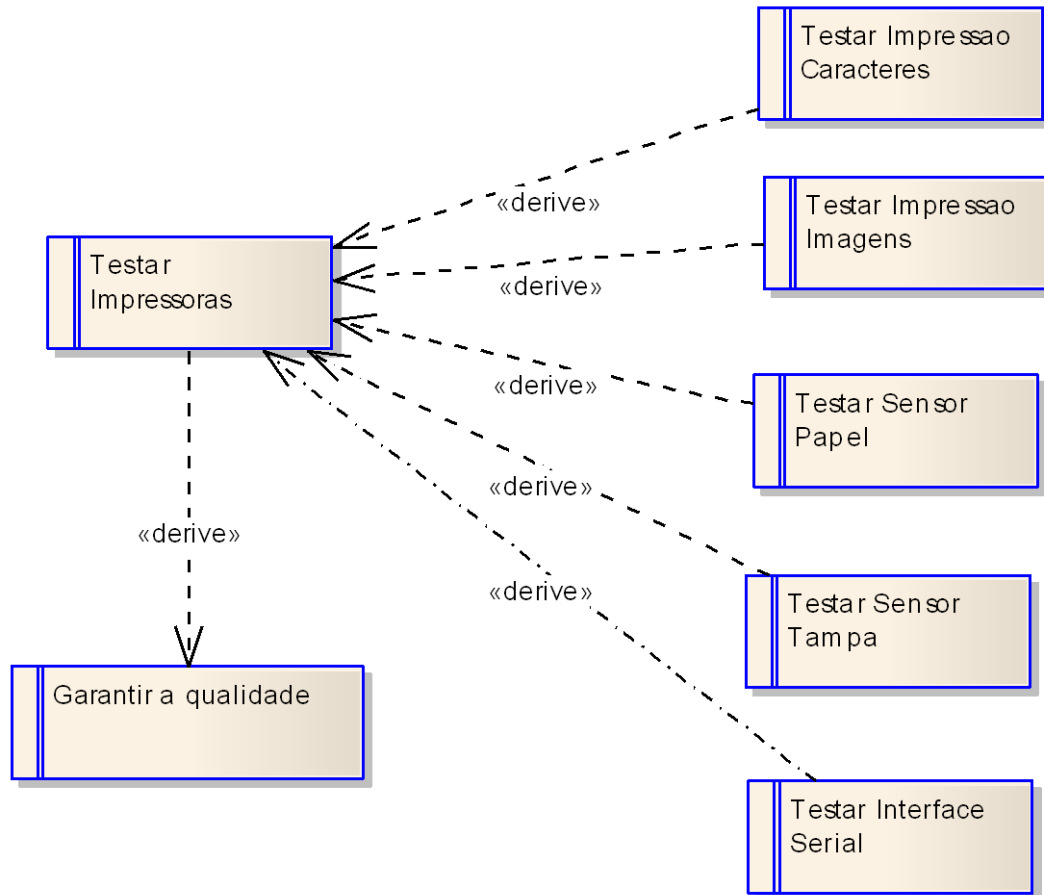


Figura 52 - Detalhamento de Necessidades, Etapa 3
Fonte: Autoria Própria

A atividade seguinte desta etapa foi detalhar os requisitos e mapear seus relacionamentos com as características. O sub-requisito 5.1 foi detalhado nos seguintes sub-requisitos essenciais “5.1.1 Testar Memória” e “5.1.2 Testar Interface Serial”.

Por sua vez, o sub-requisito 5.2 foi detalhado nos seguintes requisito essenciais: “5.2.1 Testar Sensor Papel”, “5.2.2 Testar Sensor Tampa”, “5.2.3 Testar Guilhotina”, “5.2.4 Testar Impressão Imagens” e “5.2.5 Testar Impressão Caracteres”. Os relacionamentos entre esses sub-requisitos essenciais e as características que atendem foram modelados na ferramenta EA, como ilustra a Figura 53.

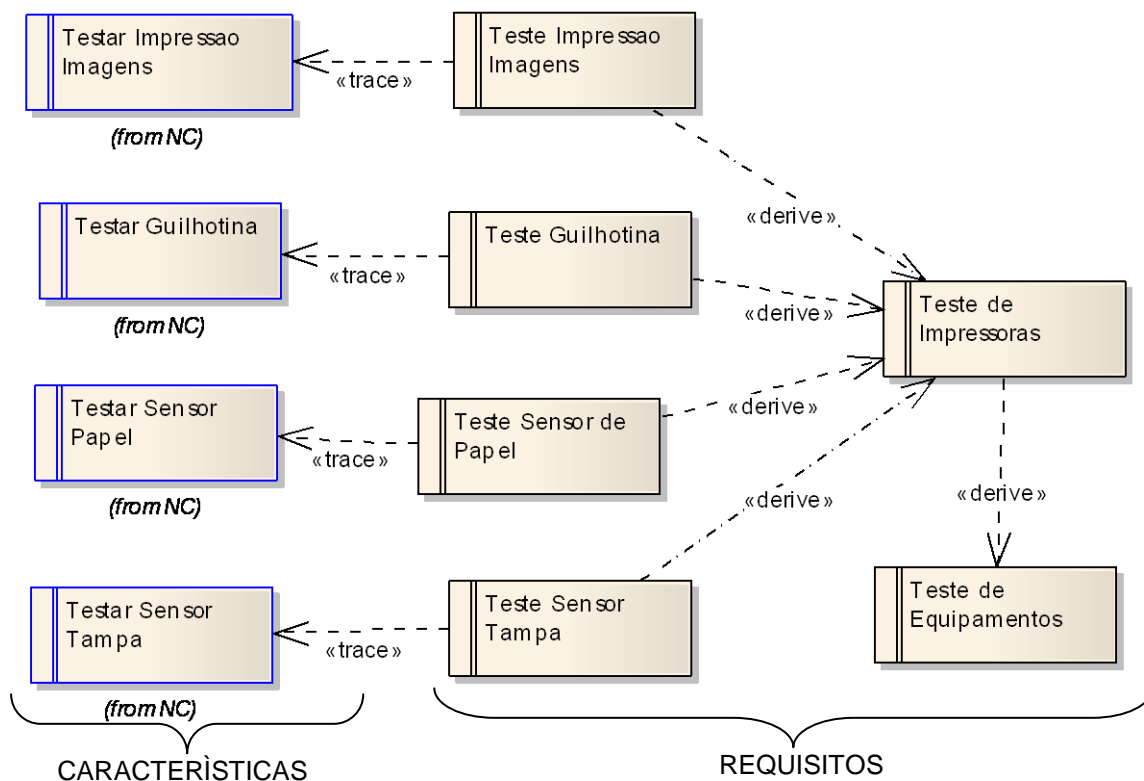


Figura 53 - Características X Sub-Requisitos Essenciais, Etapa 3
Fonte: Autoria Própria

A atividade seguinte foi mapear os relacionamentos entre Necessidades e Requisitos. A matriz obtida possui reangularidade igual a 1 e semangularidade também igual a 1, o que a torna uma matriz desacoplada como ilustra a Figura 54.

Matriz de Projeto		Reangularidade: 1 Semangularidade: 1												
Tipo de matriz: Necessidades X Requisitos		Nível: 3												
Necessidade \ Requisito	1 Atualizaçã automática do software	2 Configu de testes	3.1 Relatório Teste de Placas	3.2 Relatório Teste de Impresso	4.1 Identificar Modelo do Equipament por Numero	4.2 Emissao de Som para falha em Teste	5.1.1 Testar Memóric	5.1.2 Testar Interface Serial	5.2.1 Testar Sensor Papel	5.2.2 Testar Sensor Tampa	5.2.3 Testar Guilhotir	5.2.4 Testar Impressa Imagens	5.2.5 Testar Impressa Caracere	
1 Sistema com um mecanismo de atu...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
2 Necessita-se que os testes sejam ...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3.1 Analisar os resultados de Testes ...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3.2 Analisar os resultados de Testes ...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4.1 Identificar Modelo do Equipament...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4.2 Emitir Sons Indicativos de Result...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.1.1 Garantir que a Memória está Fu...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.1.2 Garantir que a Interface Serial F...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.2.2 Garantir que a o Sensor de Ta...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.2.1 Garantir que o Sensor de Papel...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.2.3 Garantir que a Guilhotina Funci...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5.2.4 Garantir que a Impressao de Im...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
5.2.5 Garantir que a Impressao de Ca...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Figura 54 - Matriz de Projeto, Etapa 3
Fonte: Autoria Própria

Nesta etapa também foram elaborados os casos de uso independentes relacionados a todos os sub-requisitos essenciais identificados até aqui. A Figura 55 ilustra parcialmente o diagrama de casos de uso desta etapa e seu respectivo relacionamento com os requisitos.

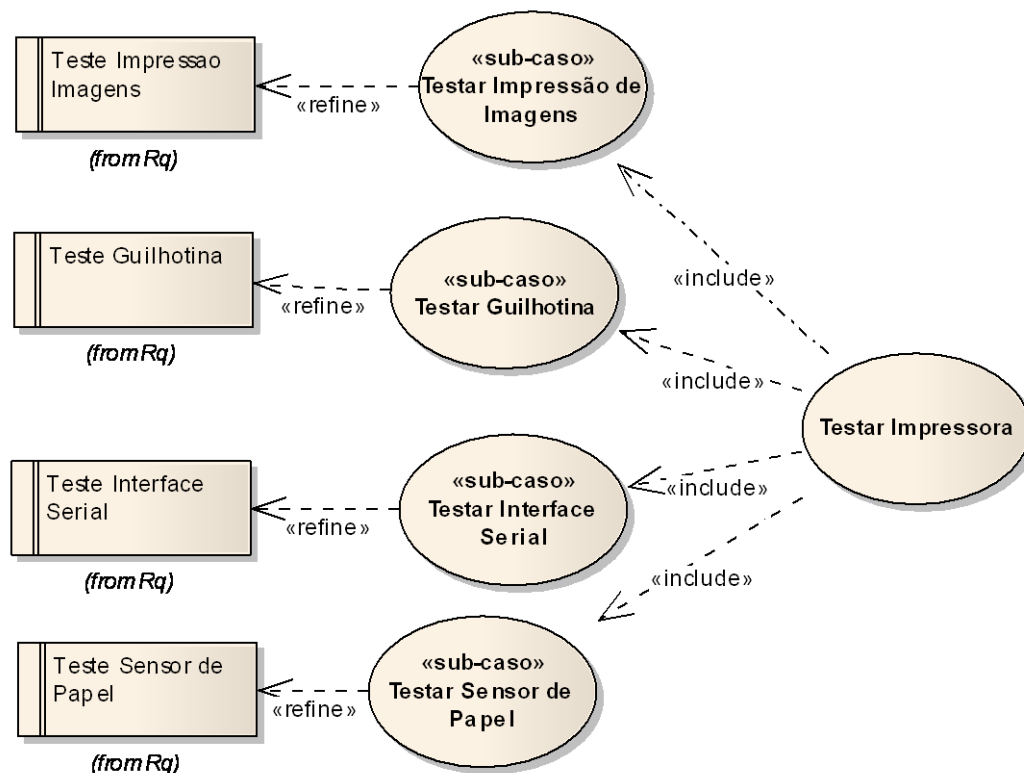


Figura 55 - Requisitos X Casos de Uso, Etapa 3
Fonte: Autoria Própria

As subcolaborações correspondentes aos casos de uso também foram identificadas nesta etapa. As subcolaborações identificadas possuem nomes idênticos aos casos de uso que realizam e foram as seguintes: “1 Atualizar Versão”, “2 Configurar Teste de Equipamento”, “3.1 Gerar Relatório de Teste de Placas”, “3.2 Gerar Relatório de Teste de Impressoras”, “4.1 Identificar modelo”, “4.2 Emitir Som Erro”, “5.1.1 Testar Memória”, “5.1.2 Testar Interface Serial”, “5.2.1 Testar Sensor de Papel”, “5.2.2 Testar Sensor de Tampa”, “5.2.3 Testar Guilhotina”, “5.2.4 Testar Impressão de Imagens” e “5.2.5 Testar Impressão de Caracteres”.

Com os sub-requisitos essenciais identificados e seu relacionamento com os casos de uso definidos encerra-se esta etapa. A próxima atividade, decomposição dos requisitos essenciais, marca o início da quarta etapa.

5.3.4 Etapa 4 – Foco nos Requisitos Técnicos

Na quarta etapa da abordagem, a primeira atividade foi decompor novamente os sub-requisitos essenciais em sub-requisitos técnicos. Nesta atividade foram identificadas algumas decomposições dos requisitos Essenciais. A modelagem destas características e seu relacionamento com os elementos dos níveis anteriores na ferramenta EA é parcialmente ilustrada na Figura 56. Os sub-requisitos técnicos apresentados na figura são: “Montar Comando de Teste de Sensor de Papel”, “Escrever Comando na Porta Serial”, “Ler Resultado do Comando” e “Interpretar Resultado do Teste de Sensor de Papel”.

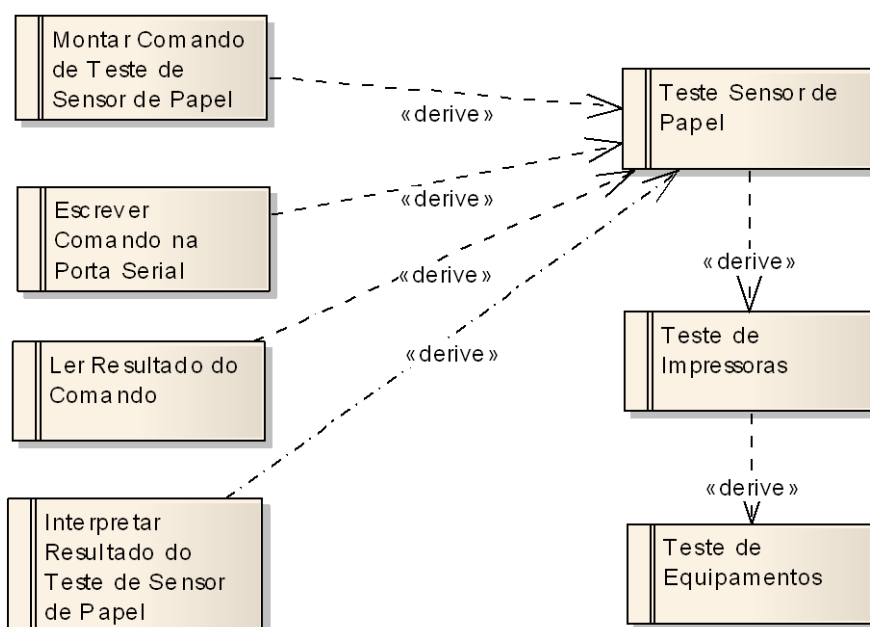


Figura 56 - Detalhamento de Requisitos, Etapa 4
Fonte: Autoria Própria

A atividade seguinte à identificação de todos os sub-requisitos técnicos é a identificação dos subcasos dependentes correspondentes a estes requisitos. A seguir os relacionamentos entre requisitos e casos de uso são mapeados, encerrando a quarta etapa desta abordagem.

A partir daí é necessário detalhar as subcolaborações e seus papéis para seguir então para a modelagem de serviços técnicos, objetos e classes do projeto. Estas atividades correspondem à abordagem proposta por Pimentel (2007) e não são apresentadas neste caso de estudo.

5.4 Conclusão

Este capítulo apresentou um caso de estudo onde foram especificados os requisitos de um sistema de teste de impressoras. Foram apresentados os objetivos do caso de estudo, a descrição do sistema e as ferramentas de *software* que apoiaram o desenvolvimento do caso de estudo. O caso de estudo foi realizado seguindo a abordagem de especificação de requisitos proposta nessa dissertação. A aplicação da abordagem seguiu as etapas e níveis de decomposição definidos.

A cada etapa foram definidos os elementos, como problemas, necessidades e requisitos. Estes elementos foram modelados por meio de uma ferramenta de modelagem de sistemas com objetivo de exemplificar a utilização da linguagem de modelagem para fins da abordagem proposta. A cada etapa também foram mapeados os relacionamentos entre elementos utilizando-se o protótipo desenvolvido para a verificação da independência funcional das matrizes de projeto. Seguindo-se a abordagem proposta, ao final do projeto é possível rastrear desde sub-requisitos técnicos até os problemas que os deram origem. Da mesma forma, é possível rastrear um requisito até os sub-requisitos de mais baixo nível que ele gerou e vice versa.

O caso de estudo exemplificou os principais aspectos a abordagem proposta a cada uma de suas etapas. Como é possível observar, este caso de estudo apresentou um cenário em que todas as matrizes são desacopladas. Este tipo de cenário pode ser considerado muito “perfeito”, pois não costuma ser muito comum. Outros estudos são necessários, portanto, para se avaliar cenários diferentes de especificação de requisitos tanto de *software* como de outros tipos de produtos. Por sua vez, os experimentos realizados apresentaram cenários mais diversificados, com matrizes que muitas vezes não eram quadradas ou apresentavam acoplamentos. Os experimentos realizados são apresentados no próximo capítulo.

6 Experimentos

Este capítulo apresenta os experimentos realizados para avaliar a abordagem proposta nesta dissertação. Como exemplo é apresentado o projeto de um *software* elaborado nos experimentos. Estes experimentos serviram para avaliar se o uso da abordagem proposta permite aplicar os princípios da Teoria de Projeto Axiomático à especificação de requisitos de um projeto de *software*. Neste capítulo, são apresentados os objetivos dos experimentos, a descrição da metodologia utilizada nos experimentos, a descrição dos problemas do ponto de vista do cliente, a descrição da primeira fase do experimento em que não foi aplicado o método proposto e, finalmente, a descrição da segunda fase, em que houve a aplicação da abordagem proposta na especificação de requisitos.

6.1 Objetivos dos Experimentos

O objetivo geral dos experimentos realizados foi avaliar se a aplicação do Projeto Axiomático nas atividades de especificação de requisitos traz resultados positivos para o projeto e, conseqüentemente, para o cliente. Além disso, deseja-se verificar se a aplicação do Axioma da Independência permite identificar melhor os requisitos adequados do projeto e garantir a satisfação das necessidades do cliente. Finalmente outro objetivo dos experimentos foi avaliar quais vantagens a aplicação da abordagem proposta traz para os analistas de requisitos e de sistemas.

6.2 Metodologia dos Experimentos

Os experimentos realizados neste trabalho consistiram em identificar problemas com ajuda de voluntários, que exerciam o papel de cliente. Todos os voluntários, exceto pelo primeiro, eram leigos no assunto de desenvolvimento de sistemas. No total foram realizados cinco experimentos. O primeiro teve como objeto um sistema editor de texto e contou com a ajuda do Prof. Dr. Paulo César Stadzis. O segundo teve como objeto um sistema de controle de estoque e contou com a ajuda da voluntária Cassiane Poersch. O terceiro teve como objeto um sistema de estudo e contou com a ajuda da voluntária Daiana Pereira. O quarto teve como objeto um

sistema de teste de roupas e contou com a ajuda da voluntária Liliane Pereira. O quinto teve como objeto um sistema de relatório de visitas e contou com a ajuda da voluntária Andréia Wrublevski. Este último experimento é apresentado em detalhes neste capítulo.

Na primeira etapa do experimento este cliente definia, de maneira geral, um problema que precisava ser solucionado. Em seguida, era solicitado a ele que definisse o que ele necessitava em um sistema que, supostamente, resolveria seus problemas. A seguir, eram especificados os requisitos do projeto sem que houvesse nenhum tipo de preocupação em criar um ou mais requisitos para cada necessidade. Até este ponto o principal objetivo era identificar os requisitos do sistema de uma maneira muito semelhante à que ocorre comumente nos projetos de *software*.

A segunda etapa do experimento consistia em refazer a especificação dos requisitos, mas desta vez o objetivo era voltar ao ponto em que o cliente definiu as necessidades e passar a aplicar o modelo proposto. Assim, a primeira atividade era questionar o cliente sobre o porquê de ele haver identificado cada uma das necessidades. Com estas respostas em mãos, os problemas e necessidades eram reformulados e seguia-se com a aplicação do modelo até que fosse obtido um conjunto de requisitos que estivesse relacionado com as necessidades atendendo ao Axioma da Independência. O objetivo final do experimento era comparar os resultados obtidos na primeira e na segunda etapa e avaliar os efeitos positivos ou negativos do uso do modelo proposto. A seguir é apresentado o exemplo de um dos experimentos realizados.

6.3 Descrição do Problema

Este caso de estudo diz respeito a uma distribuidora de livros, em que o cliente necessita um sistema de relatórios de visitas a clientes. O problema descrito pelo cliente é o seguinte:

“Em minha empresa os Gestores conhecem sua carta de clientes, mas esta informação fica somente com eles. Quando um gestor faz uma visita a um cliente não repassa para ninguém os detalhes da visita, registra no sistema da empresa somente a data e pessoa com quem conversou. Os acordos realizados com o cliente, sobre condições da compra nem sempre são repassados para a empresa. Sempre que um novo gestor entra na empresa

e assume a carteira de cliente de outro, precisa iniciar o relacionamento com o cliente a partir do zero.”

6.4 Primeira Etapa

Após o cliente explicar seu problema, foi elaborada uma lista com cada um dos problemas identificados, a lista pode ser vista na Tabela 6.

Tabela 6 - Lista de Problemas da Primeira Etapa

ID	Descrição do Problema
P1	Gestores conhecem sua carta de clientes, mas esta informação fica somente com eles
P2	Quando um gestor faz uma visita a um cliente não repassa para ninguém os detalhes da visita, apenas registra no sistema Empresa, Data e pessoa com quem conversou.
P3	Empresa não tem conhecimento dos acordos realizados com o cliente, das condições da compra.
P4	Quando um novo gestor entra na empresa e assume a carteira de cliente de outro, precisa iniciar o relacionamento com o cliente do zero.

Fonte: Autoria Própria

Na primeira etapa, por meio de uma entrevista, foram obtidas as necessidades definidas pelo cliente, que são apresentadas na Tabela 7. Nesta fase, em nenhum momento o cliente foi questionado a respeito das razões pelas quais definiu este conjunto de necessidades. O objetivo foi manter o analista livre de influências, de maneira que simplesmente definisse os requisitos do sistema da forma que estava habituado a fazer.

Tabela 7 - Lista de Necessidades da Primeira Etapa

ID	Descrição da Necessidade
N1	Necessito um cadastro com os dados completos do cliente
N2	Necessito que o sistema esteja online
N3	Necessito um controle de acesso de usuários
N4	Necessito que o sistema permita a criação de usuários com perfil de <i>Gestor</i> , que só terá acesso aos dados dos seus clientes
N5	Necessito de um perfil de usuário de Administrador que tenha acesso aos dados de todos os clientes
N6	Necessito um relatório completo de cada visita
N7	Necessito manter um histórico de todas as visitas feitas ao cliente
N8	Necessito ter um guia dos assuntos que o gestor deverá abordar na visita
N9	O sistema deverá avisar o administrador por <i>email</i> sobre o cadastro de um novo relatório.

Fonte: Autoria Própria

Após identificar os problemas e necessidades do cliente passou-se para a análise dos relacionamentos existentes entre eles. Na Figura 57, foram definidos, para cada necessidade, quais são os problemas que ela resolvia, marcando um X na

célula que liga o problema e a necessidade. Pode-se notar pela matriz que muitas necessidades não puderam ser relacionadas a nenhum dos problemas citados, formando uma matriz não quadrada.

	N1	N2	N3	N4	N5	N6	N7	N8
P1	X				X	X		
P2						X		
P3						X		
P4	X						X	

Figura 57 - Problemas X Necessidades da Primeira Etapa
Fonte: Autoria Própria

Para atender as necessidades descritas pelo cliente foram elaborados pelo analista os um conjunto de requisitos, conforme apresentado na Tabela 8.

Tabela 8 - Lista de Requisitos da Primeira Etapa

ID	Descrição do Requisito
R1	O sistema deverá ter uma funcionalidade de cadastro do cliente
R2	O sistema deverá ser desenvolvido para <i>web</i>
R3.1	O sistema deverá ter uma funcionalidade de controle de acesso de usuários
R3.2	O sistema deverá ter uma funcionalidade de cadastro de usuários
R4	O sistema deverá possuir um usuário com perfil de <i>Gestor</i> com acesso exclusivo aos dados dos clientes relacionados a ele
R5	O sistema deverá possuir um usuário com perfil de <i>Administrador</i> com acesso aos dados de todos os clientes
R6	O sistema deverá criar um relatório de visitas com todos os detalhes importantes para a empresa
R7	O sistema deverá permitir consultar o histórico de todos os relatórios de visitas
R8	O sistema deverá ter um guia dos assuntos que o gestor deverá abordar na visita
R9	O sistema deverá enviar avisos por <i>email</i> para o administrador para avisar sobre o cadastro de cada novo relatório.

Fonte: Autoria Própria

A seguir, foi criada a matriz de relacionamento entre necessidades e requisitos. Assim, foi definido, para cada requisito, quais as necessidades que ele atendia, marcando um X na célula que liga a necessidade e o requisito. A matriz obtida, apresentada na Figura 58, era semi-acoplada e, portanto, aceitável. De qualquer maneira, prosseguiu-se o experimento para observar que diferenças haveria entre o conjunto de requisitos obtidos na primeira fase em relação aos que seriam obtidos na segunda fase.

	R1	R2	R3.1	R3.2	R4	R5	R6	R7	R8	R9
N1	X									
N2		X								
N3			X	X	X	X				
N4					X					
N5						X				
N6							X			
N7							X	X		
N8									X	
N9										X

Figura 58 - Necessidades X Requisitos da Primeira Etapa
Fonte: Autoria Própria

6.5 Segunda etapa

Nesta etapa do experimento retornou-se à lista de necessidades com o objetivo de entender por que o cliente elaborou cada uma das necessidades que foram apontadas durante a conversa. A forma utilizada para entender cada necessidade simplesmente perguntando “Por que isso é necessário?”. Assim foi possível identificar problemas não identificados inicialmente, por meio da explicação do cliente de por que o sistema deveria atender cada uma das necessidades apresentadas. Isso deve ser feito por que, com frequência, o cliente não deixa explícitos muitos dos problemas que deseja solucionar. Na Tabela 9 encontram-se as respostas obtidas do cliente, no caso estudado.

Tabela 9 - Motivação das Necessidades

ID	Descrição da Necessidade e sua explicação
N1	Necessito um cadastro com os dados completos do cliente. - Por que a empresa não conhece muitos de seus potenciais clientes. - Por que constantemente os gestores esquecem dados fundamentais do cliente.
N2	Necessito que o sistema esteja online. - Por que desta forma não preciso esperar ele retornar de viagem para entregar os relatórios.
N4	Necessito que o sistema permita a criação de usuários com perfil de Gestor, que só terá acesso aos dados dos seus clientes. - Porque desejo evitar eventuais casos de sabotagem entre gestores. - Porque desejo evitar que um gestor jogue a culpa em outros por seus erros.
N6	Necessito um relatório completo de cada visita. - Porque frequentemente não recebo todas as informações sobre a visita, descontos acordados, formas de pagamento e outros dados importantes.
N8	Necessito ter um guia dos assuntos que o gestor deverá abordar na visita. - Por que eventualmente o gestor esquece algum detalhe importante na conversa.
N9	O sistema deverá avisar o administrador por <i>email</i> sobre o cadastro de um novo relatório. - Por que assim o administrador saberá imediatamente que foi cadastrado um novo relatório. (Sistema está <i>on-line</i> , portanto irá disponibilizar esta informação. Item eliminado.)

Fonte: Autoria Própria

O passo seguinte do experimento foi redefinir a lista de problemas do cliente com base nas respostas obtidas no passo anterior. Cada resposta, em geral, costuma apontar para um problema que deve ser solucionado pelo sistema. Desta forma, é possível redefinir a lista de problemas e, eventualmente, também a lista de necessidades. Em seguida, o ideal é revisar a nova lista de problemas em conjunto com o cliente para ter certeza que cada um dos problemas identificados pelos analistas de negócios faz sentido. Eventualmente, pode-se, inclusive, buscar identificar a raiz dos problemas, caso perceba-se que não estão claros o bastante. A Tabela 10 apresentada a seguir mostra como ficou a lista de problemas do sistema de Relatório de visitas a clientes.

Tabela 10 - Problemas Reformulados

ID	Descrição do Problema
P1.1	Empresa conhece mal seus clientes.
P1.2	Gerentes esquecem-se de solicitar dados importantes do cliente.
P2	Demora em receber relatórios por que gestores trabalham em locais distantes e visitam a empresa com pouca frequência.
P3	Acesso de pessoas não autorizadas ao sistema
P4.1	Risco de que as pessoas não assumam a responsabilidade pelos dados que cadastram no sistema.
P4.2	Risco de acesso indesejado a informações do sistema por pessoas que não tenham essa permissão.
P5	Problema de acesso aos dados gerados pelos gestores.
P6.1	Informações incompletas a respeito das visitas realizadas aos clientes
P6.2	Informações incompletas dos pedidos realizados pelo cliente.
P7	Gestores conhecem sua carta de clientes, mas o conhecimento fica somente com eles.
P8	Frequentemente alguns itens importantes são esquecidos pelo gestor.

Fonte: A autoria Própria

O conjunto de necessidades também foi redefinido, como ilustra a Tabela 11 a seguir.

Tabela 11 - Necessidades Reformuladas

ID	Descrição da Necessidade
N1	Necessito um cadastro com os dados completos do cliente
N2	O sistema precisa utilizar a web para transmitir os dados para o servidor da empresa
N3	O sistema precisa ter um controle de acesso de usuários
N4.1	O sistema precisa permitir o cadastro dos usuários que têm direito de acesso ao sistema
N4.2	O sistema precisa ter um perfil de Gestor, que só terá acesso aos dados dos clientes individuais do gestor
N5	O sistema precisa ter um perfil de Administrador que tenha acesso aos dados de todos os clientes
N6.1	Necessito um relatório completo de cada visita
N6.2	O sistema ter uma opção de gerar pedidos do cliente
N7	Necessito manter um histórico de todas as visitas feitas ao cliente
N8	O sistema precisa ter um guia dos assuntos que o gestor deverá abordar na visita

Fonte: A autoria Própria

A Figura 59, resultante do mapeamento entre problemas e necessidades é desacoplada, tendo somente uma questão não resolvida que é o relacionamento entre N1 e P1.1, P1.2. Segundo os princípios de análise identificados ou este problema deveria ser transformado em um só ou a necessidade N1 deveria ser reinterpretada para que fossem identificadas duas necessidades distintas. Outros relacionamentos semi-acoplados que formaram um “L”, como no caso de P3 e P4.2, são considerados aceitáveis segundo os princípios de análise estabelecidos.

	N1	N2	N3	N4.1	N4.2	N5	N6.1	N6.2	N7	N8
P1.1	X									
P1.2	X									
P2		X								
P3			X							
P4.1				X						
P4.2			x		X					
P5		x				X				
P6.1							X			x
P6.2								X		
P7									X	
P8										X

Figura 59 - Problemas X Necessidades da Segunda Etapa
Fonte: Autoria Própria

Tabela 12 - Novo Conjuntos de Requisitos

ID	Descrição do Requisito
R1	O sistema deverá ter um cadastro com os dados completos do cliente
R2	O sistema deverá utilizar a <i>web</i> para transmitir os dados para um <i>Web Server</i> da empresa
R3	O sistema deverá ter um controle de acesso de usuários
R4.1	O sistema deverá ter um cadastro dos usuários do sistema
R4.2	O sistema deverá ter um perfil de Gestor, que só terá acesso aos dados dos clientes individuais do gestor
R5	O sistema deverá ter um perfil de Administrador que tenha acesso aos dados de todos os clientes
R6.1	O sistema deverá ter uma funcionalidade de geração de relatório de visita
R6.2	O sistema deverá ter funcionalidade de envio de pedidos do cliente
R7	O sistema deverá manter um histórico de todas as visitas cadastradas no sistema
R8	O sistema deverá ter um guia dos assuntos que o gestor deverá abordar na visita

Fonte: Autoria Própria

Quando uma matriz de Problemas e Necessidades aceitável é obtida, pode-se seguir adiante, e passar a elaborar os requisitos que atenderão às necessidades encontradas. Caso contrário será necessário voltar atrás na formulação das necessidades para obter uma matriz aceitável. No caso estudado não foi necessário

reformular as necessidades por haver sido formada uma matriz semiacoplada, por este motivo passou-se para a elaboração dos requisitos do sistema. A Tabela 12 apresenta uma lista de requisitos elaborados com o objetivo de atender as necessidades do cliente do Sistema de Relatório de Visitas.

A última atividade realizada neste experimento foi o mapeamento dos relacionamentos entre as Necessidades e os Requisitos identificados. O objetivo de elaborar esta matriz é avaliar se ela atende ao axioma da independência definido. Em outras palavras, o que se espera em uma matriz como esta, do ponto de vista dos relacionamentos, é obter sempre uma matriz desacoplada ou semiacoplada. Uma matriz que apresente acoplamento entre Necessidades e Requisitos aponta para problemas de interpretação destas Necessidades e Requisitos e seus relacionamentos. Da mesma forma que na matriz de relacionamento entre problemas e necessidades, espera-se que a matriz seja sempre quadrada, ou seja, que exista o mesmo número de Necessidades e Requisitos. Uma matriz não quadrada indicaria a possibilidade de estarem faltando ou sobrando Requisitos. Como é possível ver na Figura 60 a matriz resultante no caso de estudo apresentado é uma matriz quadrada e desacoplada.

	R1	R2	R3	R4.1	R4.2	R5	R6.1	R6.2	R7	R8
N1	X									
N2		X								
N3			X							
N4.1				X						
N4.2					X					
N5						X				
N6.1							X			
N6.2								X		
N7									X	
N8										X

Figura 60 - Necessidades X Requisitos da Segunda Etapa
Fonte: Autoria Própria

6.6 Conclusões

Foi possível perceber, por meio deste experimento, que o projeto sofreu alguns refinamentos da segunda vez que foi executado. Na primeira vez o analista

apenas anotou os problemas citados pelo cliente e as necessidades que ele dizia ver no sistema. Os requisitos foram elaborados sem que houvesse maior preocupação em entender o que cada uma das necessidades representava efetivamente para o cliente. Por outro lado, na segunda vez o analista quis saber o porquê de cada necessidade existir. Isso o levou a identificar um número maior de problemas do que aqueles que o cliente havia citado da primeira vez, passando de quatro para onze problemas. Em consequência disso foi possível identificar novos requisitos e eliminar outros identificados na primeira fase. Desta forma, os requisitos obtidos na segunda fase do estudo foram mais consistentes. Isso leva a crer que a segunda solução atenderia de forma mais completa as necessidades do cliente, caso fosse *implementada*.

7 Conclusões e Trabalhos Futuros

7.1 Conclusões

Nessa dissertação foi apresentada uma abordagem de especificação de requisitos baseada em Projeto Axiomático que pode ser integrada a um processo de desenvolvimento de *software* como o Processo Unificado. A abordagem proposta nesta dissertação fornece um método para auxiliar os analistas na definição de requisitos adequados para projetos de sistemas de *software*. Este método se baseia nos princípios gerais de bons projetos que fazem parte da Teoria de Projeto Axiomático.

A abordagem proposta oferece as seguintes vantagens para especificação de requisitos de sistemas de *software*:

- Fornecer um método de especificação de requisitos com atividades e etapas bem definidas;
- Fornecer critérios para avaliar a qualidade do conjunto de problemas, necessidades e requisitos definidos pelo analista;
- Auxiliar a manter uma alta probabilidade de que o sistema desenvolvido atenda as necessidades do cliente;
- Favorecer a rastreabilidade entre artefatos que *implementam* o sistema e os requisitos de alto nível, bem como os problemas que deram origem a estes requisitos;

O entendimento do negócio do cliente, assim como de seus problemas e necessidades, é fundamental para uma boa especificação de requisitos. A qualidade dos requisitos costuma ser determinante para a realização de um bom projeto. Da mesma forma, a qualidade do projeto é fundamental para a realização de um produto de qualidade. Por outro lado, é comum ocorrerem problemas com a especificação de requisitos. A existência desse tipo de problemas motivou a elaboração da abordagem de especificação de requisitos proposta nesta dissertação.

Com vistas a facilitar sua aplicação em projetos de sistemas de *software*, foram estabelecidas analogias entre a abordagem proposta e o Processo Unificado. Da mesma forma, foram estabelecidas analogias entre conceitos da abordagem

proposta e elementos de modelagem das linguagens UML e SysML com a finalidade de facilitar a modelagem de requisitos de sistemas de *software*. Os conceitos da abordagem foram definidos para permitir diferentes níveis de decomposição, com objetivo de permitir a aplicação de um processo em *zig-zag* como o proposto na Teoria de Projeto Axiomático.

Foram definidas quatro etapas para a abordagem de especificação de requisitos proposta para facilitar a aplicação do processo de decomposição em *zig-zag*. Cada uma destas etapas possui um foco específico, a primeira foca nos problemas, a segunda nas necessidades, a terceira nos requisitos essenciais e a quarta nos requisitos técnicos. Desta forma, a abordagem pode ser aplicada nas fases de concepção e elaboração do Processo Unificado, porém mudando o foco a cada etapa.

A matriz de projeto é uma ferramenta importante na abordagem proposta nessa dissertação uma vez que permite avaliar, do ponto de vista da Teoria de Projeto Axiomático, a qualidade dos requisitos especificados. Além disso, a matriz de projeto tem um papel fundamental na rastreabilidade dos requisitos ao longo do processo de desenvolvimento do sistema.

Um caso de estudo de um sistema de teste de equipamentos foi desenvolvido com a finalidade de exemplificar a aplicação da abordagem proposta. A aplicação da abordagem seguiu as etapas estabelecidas, apresentando exemplos relativos à execução das principais atividades de cada etapa. Este caso de estudo ilustra como a decomposição funcional e o *zig-zagueamento* podem ser aplicados na especificação de requisitos de um sistema de *software*.

Durante o projeto de pesquisa foram realizados diversos experimentos, cujos resultados foram apresentados nesta dissertação. Estes experimentos contribuíram para a avaliação das atividades propostas pela abordagem bem como para a avaliação dos benefícios da abordagem em si. Da mesma forma, os experimentos contribuíram para o estabelecimento de critérios de avaliação das matrizes de projeto.

Outra contribuição deste trabalho foi o desenvolvimento de um protótipo de uma ferramenta para avaliar a coerência das matrizes de projeto em relação ao Axioma da Independência. Esta ferramenta permite validar todas as matrizes obtidas durante aplicação da abordagem proposta em seus diferentes níveis de decomposição.

Durante os estudos relativos à abordagem proposta também se observou o seguinte:

- Mapear itens de dois domínios em uma matriz, com objetivo de avaliá-la do ponto de vista do Axioma da Independência, pode auxiliar na identificação de inconsistências na interpretação dos elementos de cada domínio. Eventualmente, essa análise pode ajudar a identificar elementos que estão faltando ou sobrando e, assim, levar a melhorias no projeto;
- O detalhamento dos problemas e/ou desejos que levaram o cliente a tomar a decisão de adquirir um produto permite que os Engenheiros de Sistemas tenham uma visão ampliada do domínio do problema e, desta maneira, possam propor soluções alternativas. É provável que estas soluções sejam mais eficientes do que aquelas imaginadas pelo cliente e expressas por meio das características e necessidades;
- A utilização da matriz de projeto para mapear os relacionamentos entre os diversos domínios ajuda a manter a consistência e a rastreabilidade de problemas, necessidades, requisitos e elementos de arquitetura da solução;
- Este modelo pode ser aplicado ao processo de Engenharia de Requisitos em qualquer área.

7.2 Trabalhos futuros

A abordagem proposta nesta dissertação aplica a Teoria de Projeto Axiomático à atividade de especificação de requisitos. A abordagem proposta apresenta alguns elementos de linguagem de modelagem para serem utilizados na modelagem de requisitos, porém os elementos desta linguagem não foram estudados rigorosamente. Por esta razão, uma proposta de trabalhos futuros é desenvolver um perfil UML específico para a modelagem de requisitos seguindo a abordagem proposta.

Além disso, uma proposta bastante promissora de trabalhos futuros é desenvolver novos estudos para avaliar a eficiência da abordagem proposta para a especificação de requisitos de maneira mais aprofundada. Embora tenham sido realizados estudos de caso e experimentos, não foi possível avaliar quantitativa ou estatisticamente a eficiência desta abordagem. Um estudo focado na avaliação da

abordagem poderia deixar mais claras as vantagens e resultados obtidos por meio da sua aplicação.

Como foi visto no caso de estudo, foram utilizadas concomitantemente duas ferramentas, o *Enterprise Architect (EA)* e o protótipo de avaliação de matrizes. Para facilitar a aplicação da abordagem uma proposta de trabalhos futuros é a evolução do protótipo desenvolvido. Uma versão melhorada da ferramenta poderia destacar os casos correspondentes aos princípios de análise identificados, calcular o conteúdo da informação, incrementar a rastreabilidade e, até mesmo, incluir a modelagem em linguagem UML/SysML. Outra possibilidade seria desenvolver um *plug-in* para EA que seja capaz de calcular a independência funcional das matrizes de rastreabilidade e o conteúdo da informação do projeto.

Ainda em relação aos princípios de análise identificados, em um trabalho futuro poderia ser desenvolvida uma equação para avaliar a matriz levando em conta estes princípios.

Outra proposta de trabalhos futuros que pode ser bastante promissora é estudar a possibilidade de se desenvolver um método de avaliação multidimensional das matrizes de projeto. Desta forma, seria possível avaliar os relacionamentos entre três ou mais domínios ao mesmo tempo.

Referências

- ANDA, B., et. al. (2001). Estimating software development effort based on use cases - experiences from industry. *The Unified Modeling Language. 4th International Conference*. Toronto, Canadá.
- BALMELLI, L. (2007). An Overview of the Systems Modeling Language for Products and Systems. *Journal of Object Technology* , pp. 149-177.
- BERGAMINI, C. W. (1997). *Motivação nas organizações* (4 Ed ed.). São Paulo: Atlas.
- BOOCH, G., RUMBAUGH, J.; JACOBSON, I. (2006). *UML – Guia do Usuário* (2 ed.).
- CAMBRIDGE, U. (2011). Acesso em 08 de Mai de 2011, disponível em Cambridge Dictionaries Online: <http://dictionary.cambridge.org/>
- CARVER, C. S.; SCHEIER, M. F. (2000). *Perspectives on Personality*. Boston: Allyn and Bacon.
- CHIAVENATO, I. (2005). *Gerenciando com pessoas: transformando o executivo em um excelente gestor de pessoas*. São Paulo: Elsevier.
- CHIDAMBER, S. R.; KEMERER, C. F. (1994). A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering, New York, EUA, v. SE-20, n. 6* , pp. p. 476-493.
- CHRISSIS, M. B., KONRAD, M.; SHRUM, S. (2007). *CMMI - Guidelines for process integration and product improvement*. Boston: Addison Wesley.
- DARDENNE, A., van LAMSWEERDE, A.; FICKAS, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming* , pp. Volume 20, Issues 1-2, Pages 3-50.
- DAVIS, A. M. (1993). *Software Requirements: Objects, Functions and States*. Prentice-Hall.
- DEO, H. V.; SUH, N. P. (2004). Axiomatic design of customizable automotive suspension. *Anais do ICAD2004 - the 3rd International Conference on Axiomatic Design*. Seul, Coreia.
- DIJKSTRA, E. W. (1972). The humble programmer. *ACM* , 859–866.
- DO, S. H. (2004). Software product lifecycle management using axiomatic approach. *Anais do ICAD2004 - the 3rd International Conference on Axiomatic Design*. Seul, Coreia.
- DO, S. H.; PARK, G. J. (1996). Application of Design Axioms for Glass-Bulb design and Software. *Anais do CIRP Workshop on Design and Inteligent*. Tokio, Japão.

DO, S. H.; SUH, N. P. (1999). Systematic OO programming with axiomatic design. *IEEE Computer*, pp. p. 121-124.

DOMINGUEZ, J. (2009). Acesso em 14 de Junho de 2010, disponível em ProjectSmart: <http://www.projectsmart.co.uk/pdf/the-curious-case-of-the-chaos-report-2009.pdf>

DORFMAN, M.; THAYER, R. H. (1990). *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press.

ELLIS, K. (2009). *Business Analysis Benchmark : The Path to Success*. New Castle: IAG Consultin.

ELLIS, K. (2008). *Business Analysis Benchmark: The impact of Business Requirements on the Success of Technology Projects*. New Castle: IAG Consulting.

FUENTES, L.; VALLENCILLO, A. (2004). An Introduction to UML Profiles. *The European Journal for the Informatics Professional*, vol. 5, nº2 .

FUGGETTA, A. (2000). *Software Process: A Roadmap*. Acesso em 25 de Maio de 2010, disponível em CiteseerX: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=ECA6008EB879D5CE65B942DF2FFCC989?doi=10.1.1.5.2549&rep=rep1&type=pdf>

HALL, J. G., et. al. (2002). Relating Software Requirements and Architectures using Problem Frames. *Proceedings of IEEE International Requirements Engineering Conference*.

HAUSE, M. (2006). The SysML Modelling Language. *Fifth European Systems Engineering Conference*. INCOSE.

HEAVEN, W.; FINKELSTEIN, A. (2004). A UML profile to support requirements engineering with KAOS. *Software, IEE Proceedings*, 10 - 27.

HERNANDES, A. (2009). *Necessidades X Desejos*. Acesso em 08 de mai de 2011, disponível em Administradores.com.br: <http://www.administradores.com.br/informe-se/artigos/necessidades-x-desejos/30811/>

HEUMANN, J. (2003). *The Five Levels of Requirements Management Maturity*. Rational Software.

IEEE. (1997). *IEEE Standard for Developing Software Life Cycle Processes*. New York, NY: The Institute of Electrical and Electronics Engineers.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)*. New York, NY: The Institute of Electrical and Electronics Engineers.

INCOSE. (2004). *Systems Engineering Handbook - A "What To" Guide For All Se Practitioners. International Council on Systems Engineering, INCOSE - TP-2003-016-02, Version 2a* .

- ISHIKAWA, K. (1990). *Introduction to Quality Control*. Tokio: 3A Corporation.
- ISO/IEC. (2005). *ISO/IEC 15001:2005 Information technology -- Open Distributed Processing -- Unified Modeling Language (UML) Version 1.4.2*. ISO/IEC.
- JACKSON, M. (1999). Problem Analysis and Structure. *Keynote Talk at ITG/SEV Symposium*. Zürich, Switzerland.
- JACKSON, M. (1995). Problems & Requirements. *Proceedings of the IEEE Second International Symposium on Requirements Engineering* (pp. 2-8). ACM Press.
- JACOBSON, I., BOOCH, G.; RUMBAUGH, J. (1999). *Unified Software Development Process*. Addison-Wesley.
- KIM, S. J., SUH, N. P.; KIM, S. K. (1992). Design of software systems based on axiomatic design. *Anais do CIRP General Assembly*.
- KOTLER, P.; KELLER, K. L. (2006). *Administração de Marketing* (12 Ed. ed.). São Paulo: Pearson Prentice Hall.
- KRUCHTEN, P. (2003). *Introdução ao RUP - Rational Unified Process*. Rio de Janeiro: Ciência Moderna.
- KULAK, O., DURMUSOGLU, M. B.; TUFEKI, S. (2005). A complete cellular manufacturing system design methodology based on axiomatic design principles. *Journal Computers and Industrial Engineering*, Volume 48 Issue 4.
- LEE, T.; JEZIOREK, P. N. (2006). Understanding the Value of Eliminating An Off-Diagonal Term in a Design Matrix. *Proceedings of ICAD 2006*. Firenze.
- LEFFINGWELL, D.; WIDRIG, D. (2003). *Managing Software Requirements*. Boston: Addison Wesley.
- LI, Z. (2007). Progressing Problems from Requirements to Specifications in Problem Frames. London, United Kingdom: Department of Computing - The Open University.
- MASLOW, A. H. (1970). *Motivation and Personality* (2 Ed ed.). Upper Saddle River: Prentice Hall.
- MCEWEN, S. (2004). *Requirements: An introduction*. Acesso em 10 de Oct de 2010, disponível em IBM developerWorks: <http://www.ibm.com/developerworks/rational/library/4166.html>
- MORAES, A. M. (2004). *Introdução à Administração*. São Paulo: Prentice Hall.
- MURRAY, H. A. (1938). *Explorations in personality*. New York: Oxford University Press.
- MYLOPOULOS, J., CHUNG, J.; NIXON, B. (1992). Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, pp. 483-497.

NASA STI, Program. (2007). *NASA Systems Engineering Handbook*. Washington, D.C.

NUSEIBEH, B.; EASTERBROOK, S. (2000). *Requirements Engineering: A Roadmap. Proceedings of International Conference on Software Engineering (ICSE-2000)*. Limerick: ACM Press.

OLEWNIK, A. T.; LEWIS, K. E. (2003). *On validating design decision methodologies. Design Theory and Methodology Conference - DETC 2003*. Chicago, IL, EUA.

OMG. (2007). *A UML Profile for MARTE, Beta 1*. Acesso em 15 de May de 2011, disponível em OMG.org: <http://www.omg.org/cgi-bin/doc?ptc/2007-08-04.pdf>

OMG. (2008). *OMG Systems Modeling Language - Versão 1.1*. Acesso em 10 de Dez de 2010, disponível em OMG.org: <http://www.omg.org/spec/SysML/1.1>

OMG. (2010). *OMG Systems Modeling Language - Versão 1.2*.

OMG. (2010). *OMG Unified Modeling Language - Infrastructure Versão 2.3*. Nedham, Massachusetts, EUA.

PAGE-JONES, M. (1988). *The Practical Guide to Structured Systems Design*. Yourdon Press.

PIMENTEL, A. R. (2007). *Uma abordagem para projeto de software orientado a objetos baseado na teoria de projeto axiomático*. 2007. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Programa de Pós Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba, 2007

PIMENTEL, A. R.; STADZISZ, P. C. (2006). *A use case based object-oriented software design approach using the axiomatic design theory. Anais do ICAD2006 - the 4th International Conference on Axiomatic Design*. Florença, Itália.

PIMENTEL, A. R.; STADZISZ, P. C. (mar de 2006). *Application of the independence axiom on the design of object-oriented software using the axiomatic design theory. Journal of Integrated Design and Process Science*, 10, pp. 57-70.

PRESSMAN, R. S. (2006). *Engenharia de Software* (6 ed.). São Paulo, SP: Mc. Graw Hill.

RATIONAL SOFTWARE. (2003). *Rational Unified Process*. Acesso em 06 de Junho de 2010, disponível em Wthreex: <http://www.wthreex.com/rup/portugues/index.htm>

RESPECT-IT. (2007). *A KAOS Tutorial*. Acesso em 15 de Oct. de 2010, disponível em [Objectiver.com: http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf](http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf)

SCHWABER, K. (2004). *Agile project management with Scrum* (1 ed.). Microsoft Press.

SOARES, M. S.; VRANCKEN, J. (2008). Model-Driven User Requirements Specification using SysML. *JOURNAL OF SOFTWARE* , pp. 57-68.

Softex. (2011). *MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral*. Campinas, BR: SOFTEX.

STEVENS, W., MYERS, G.; CONSTANTINE, L. (1974). Structured Design. *IBM Systems Journal* , pp. pág. 115-139.

SUH, N. P. (2001). *Axiomatic Design: advances and applications*. New York, EUA: Oxford University Press.

SUH, N. P. (1990). *The Principles of Design*. New York: Oxford University Pres.

SUH, N. P.; DO, S. H. (2000). Axiomatic design of software systems. *CIRP annals*. Sidney, Australia.

THE STANDISH GROUP. (2009). *The CHAOS Report (2009)*. Boston, MA, USA: The Standish Group International, Inc.

van LAMSWEERDE, A. (2001). Goal-oriented requirements engineering: a guided tour. *Fifth IEEE International Symposium on Requirements Engineering* (pp. 249 - 262). Toronto: IEEE.

van LAMSWEERDE, A. (2004). Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. *Proceedings of the 12th IEEE International Conference on Requirements Engineering* (pp. 4 - 7). IEEE.

van LAMSWEERDE, A.; WILLEMET, L. (1998). Inferring declarative requirements specifications from operational scenarios. *IEEE Trans. on Software Engineering*, 24(12) , pp. 1089–1114.

van LAMSWEERDE, A., DARIMONT, R.; MASSONET, P. (1995). Goal- Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. *Proc. RE'95 - 2nd Intl. IEEE Symp. on Requirements Engineering* (pp. 194-203). IEEE.

VAZQUEZ, C. E., SIMÕES, G. S.; ALBERT, R. M. (2003). *Análise de Pontos de Função: medição, estimativas e gerenciamento de projetos de software*. São Paulo, Brasil: Érica.

WEILKIENS, T. (2007). *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann.

Apêndice A - Protótipo da Ferramenta

Neste anexo é apresentado o protótipo da ferramenta de avaliação de matrizes de projeto que foi desenvolvido durante este trabalho de mestrado.

Informações Técnicas sobre o protótipo

O protótipo foi desenvolvido e testado no seguinte ambiente:

Sistema Operacional: Windows 7 e Windows XP

Linguagem de programação: C# com Framework .Net 4

Ferramenta: Microsoft Visual C# Express 2010

Banco de Dados: Microsoft Access 2007

Avaliação das Matrizes

A avaliação das matrizes toma por base a reangularidade e a semangularidade calculadas por meio das equações (1) e (2) apresentadas em Pimentel (2007). Ambas, possuem um valor máximo de 1, que corresponde a um projeto desacoplado (ideal) e, desta forma, um valor 0 para estas equações corresponde um projeto acoplado (inaceitável). Quando a independência funcional diminui, o valor de “reangularidade” e de “semangularidade” decresce.

$$S = \prod_{j=1}^n \left(\frac{|A_{jj}|}{\left(\sum_{k=1}^n A_{kj}^2 \right)^{1/2}} \right) \quad (1)$$

$$R = \prod_{\substack{i=1, n-1 \\ j=1+i, n}} \left(1 - \frac{\left(\sum_{k=1}^n A_{ki} A_{kj} \right)^2}{\left(\sum_{k=1}^n A_{ki}^2 \right) \left(\sum_{k=1}^n A_{kj}^2 \right)} \right)^{1/2} \quad (2)$$

Ambas as equações são calculadas sobre todos os elementos da matriz (A_{ij}) levando em conta o peso de cada elemento. Neste trabalho os elementos da matriz podem ter peso 1 (quando existe o relacionamento entre linha e coluna) ou 0 (não existe relacionamento entre linha e coluna).

A seguir são apresentados os códigos fonte do cálculo das equações da semangularidade e reangularidade na ferramenta. Cada um dos métodos realiza seu cálculo sobre uma matriz de inteiros que corresponde à matriz exibida na tela no momento do cálculo.

Método que calcula a semangularidade

```
public static double semangularidade()
{
    double semang = 1;

    for (int j = 0; j < (matriz[0].Length - 1); j++)
    {
        double res1 = 0;
        double res2 = 0;

        for (int k = 0; k < matriz.Length; k++)
        {
            if (k == j)
                res1 = Math.Abs(matriz[k][j]);

            res2 += matriz[k][j] * matriz[k][j];
        }
        res2 = Math.Sqrt(res2);

        double res3 = 1;
        if (res2 > 0)
            res3 = (res1 / res2);

        semang = semang * res3;
    }
    return Math.Round(semang, 5);
}
```

Método que calcula a reangularidade

```

public static double reangularidade()
{
    double reang = 1;
    int j;

    for (int i = 0; i < (matriz[0].Length-1); i++)
    {
        double res1 = 0;
        double res2 = 0;
        double res3 = 0;

        j = 1 + i;
        for (int k = 0; k < matriz.Length; k++)
        {
            res1 += matriz[k][i] * matriz[k][j];
            res2 += matriz[k][i] * matriz[k][i];
            res3 += matriz[k][j] * matriz[k][j];
        }
        res1 = res1 * res1;
        res2 = res2 * res2;

        res3 = 0;
        if(res2 > 0)
            res3 = (res1 / res2);

        reang = reang * Math.Sqrt(1 - res3);
    }
    return Math.Round(reang,5);
}

```

A Figura 61 ilustra a forma como são apresentados os resultados do cálculo da reangularidade e da semangularidade na ferramenta desenvolvida.

Nec \ Req	1 Controlar iluminação a distância	2 Controle de temperatura ambiente a distância	3 Controle de som ambiente a distância	4 Controle de sistema de irrigação a distância	5 Controle de aquecimento de água a distância	6 Mecanismo de controle de acesso de pessoas
1 Controlar iluminação da casa por controle remoto	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2 Controlar temperatura da casa por controle remoto	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3 Controlar Som Ambiente por controle remoto	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4 Controlar irrigação de jardim por controle remoto	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5 Controlar aquecimento de água por controle remoto	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6 Controle de acesso de pessoas à casa por sensor	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figura 61 - Cálculo da Reangularidade e Semangularidade
 Fonte: Autoria própria

Manual de Uso

Ao abrir a ferramenta é apresentada a tela inicial. Nesta tela inicial o usuário necessita escolher o projeto com o qual deseja trabalhar. O usuário pode escolher editar um projeto cadastrado anteriormente ou criar um novo projeto, como ilustra a Figura 62. Caso o usuário queira excluir um projeto deve selecioná-lo, como se fosse editar, e clicar no botão “Excluir”.

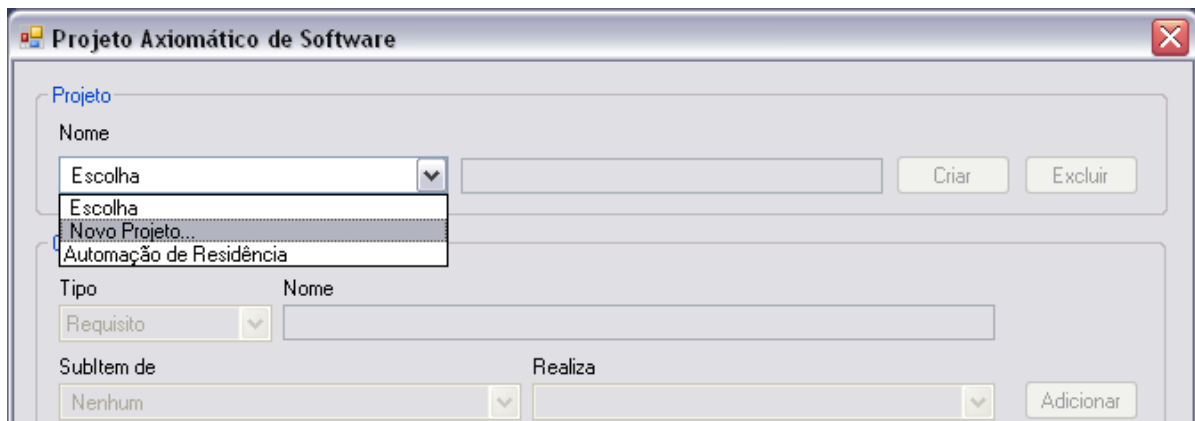


Figura 62 - Tela Inicial
Fonte: Autoria própria

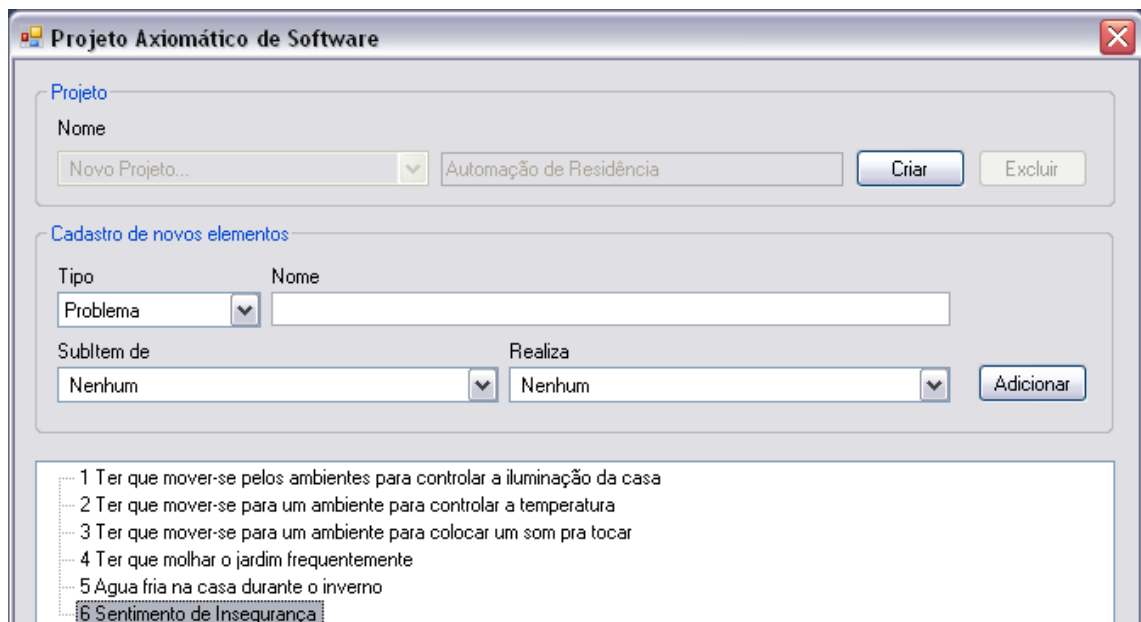


Figura 63 - Cadastrar Elemento
Fonte: Autoria própria

Após selecionar a opção desejada e clicar no botão “Criar” ou “Abrir”, o usuário pode iniciar o cadastro de elementos. Para cadastrar elementos é necessário selecionar o tipo. Cada tipo representa um dos domínios de projeto da abordagem proposta.

O usuário seleciona um tipo, como no exemplo problema, preenche as informações relativas ao problema e clica em “Adicionar”. Todos os elementos do tipo selecionado são apresentados em uma lista conforme mostra a Figura 63.

Os campos Tipo e Nome são obrigatórios no cadastro de elementos do projeto. Caso o usuário tente adicionar um elemento sem informar um destes dados é apresentada uma mensagem de alerta como ilustra a Figura 64.

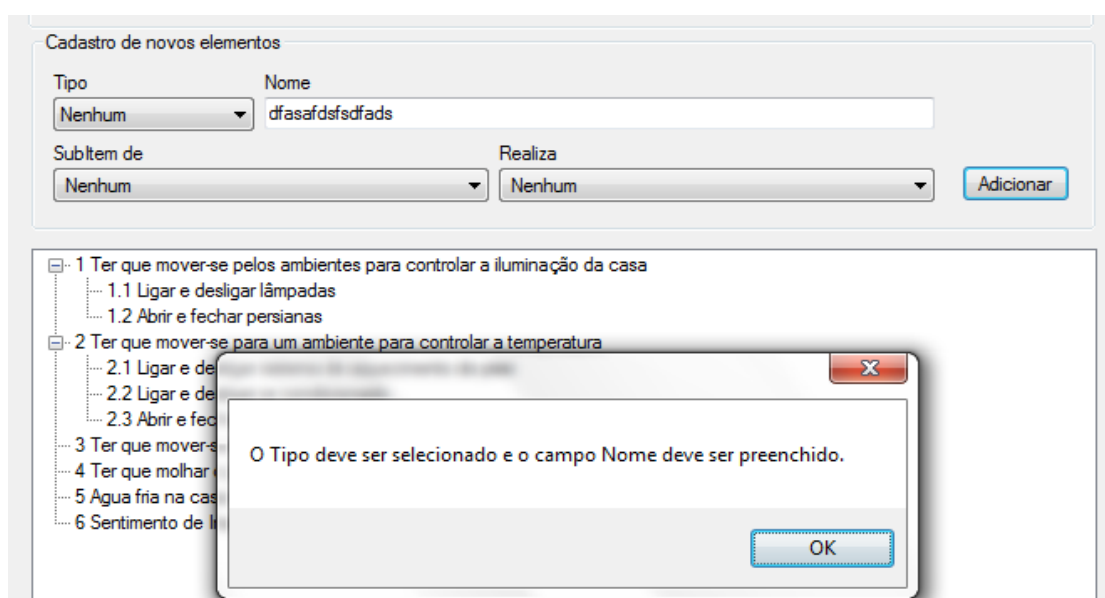


Figura 64 - Mensagem de Alerta
Fonte: Autoria própria

A Figura 65 apresenta um exemplo de inclusão de um elemento do tipo necessidade ao projeto. Neste exemplo além do tipo e do nome são definidos os campos “Subitem de” e “Realiza”. Ao selecionar um elemento da lista “Subitem de” o usuário está definindo que o novo elemento foi obtido a partir da decomposição do elemento de mais alto nível indicado no campo “Subitem de”. Ao selecionar um elemento da lista “Realiza” o usuário está definindo que o novo elemento está relacionado à um elemento de outro domínio. De acordo com o domínio selecionado é determinado o tipo do domínio realizado. No exemplo como está sendo cadastrada

uma Necessidade o elemento realizado é um Problema. Sempre que é indicado o elemento realizado é criado um relacionamento na matriz de projeto.

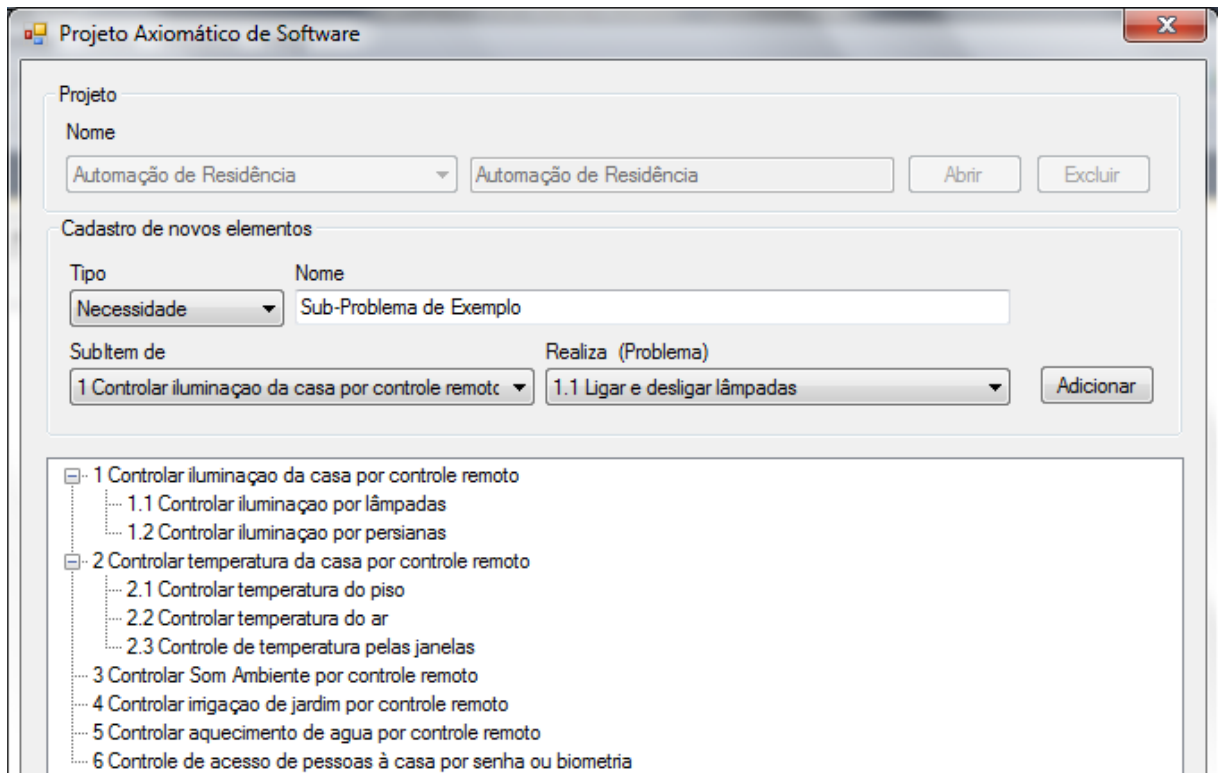


Figura 65 - Adicionar Elemento com Relacionamentos
Fonte: Autoria própria

Após cadastrar elementos de dois domínios é possível visualizar a matriz de projeto formada clicando no botão “Matrizes”, que fica na parte inferior da tela principal. A tela exibida é ilustrada na Figura 66. Nesta tela é possível visualizar todas as matrizes do projeto. Para isso é necessário selecionar primeiro o tipo de matriz e depois o nível de decomposição que se deseja visualizar. No exemplo da figura foi selecionado o tipo “Problemas X Necessidades” e o nível “2”. Neste caso são exibidos todos os elementos de nível 2 para os itens que possuem sub-divisões. Os elementos que não possuem sub-divisões são exibidos no nível anterior para que a matriz seja completa.

O usuário pode adicionar e remover relacionamentos da matriz simplesmente marcando ou desmarcando as células da matriz que deseja alterar. Ao fechar a tela os relacionamentos modificados são mantidos em memória para serem gravados com o projeto. Para salvar um projeto o usuário deve clicar em

“Gravar Projeto” ou “Alterar Projeto” na parte inferior da tela. Para fechar um projeto e selecionar outro é necessário clicar em “Fechar Projeto”.

Pro \ Nec	1.1 Controlar iluminação por lâmpadas	1.2 Controlar iluminação por persianas	2.1 Controlar temperatura do piso	2.2 Controlar temperatura do ar	2.3 Controle de temperatura pelas janelas	3 Controlar Som Ambiente por controle remoto	4 Controlar irrigação de jardim por controle remoto	5 Controlar aquecimento de água por controle remoto	6 Controle de acesso de pessoas à casa por senha ou biometria
1.1 Ligar e desligar lâmpadas	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1.2 Abrir e fechar persianas	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.1 Ligar e desligar sistema de aquecimento do piso	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.2 Ligar e desligar ar condicionado	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.3 Abrir e fechar janelas	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3 Ter que mover-se para um ambiente para colocar...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4 Ter que molhar o jardim frequentemente	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5 Água fria na casa durante o inverno	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6 Sentimento de Insegurança	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figura 66 - Matriz de Projeto
Fonte: Autoria própria

Anexo A – Teoremas e Corolários Relacionados com os Axiomas

Neste anexo são apresentados corolários e teoremas relacionados com os axiomas 1 e 2. Os corolários e teoremas apresentados a seguir foram traduzidos de (SUH, 2001).

A.1 Corolários

Corolário 1 (Desacoplamento de projetos acoplados) Desacople ou separe as partes ou aspectos de uma solução se os requisitos funcionais estiverem acoplados ou se tornarem interdependentes no projeto proposto.

Corolário 2 (Minimização dos requisitos funcionais) Minimize o número de requisitos funcionais e restrições.

Corolário 3 (Integração das partes físicas) Integre características de projeto em uma única parte física se os requisitos funcionais puderem ser satisfeitos independentemente na solução proposta.

Corolário 4 (Uso de padronização) Use partes padronizadas ou intercambiáveis se o uso destas partes estiver consistente com os requisitos funcionais e restrições.

Corolário 5 (Uso de simetria) Use formas e componentes simétricos se eles estiverem consistentes com os requisitos funcionais e restrições.

Corolário 6 (Maior faixa de variação de projeto) Especifique a maior faixa de variação de projeto permitido quando estiver estabelecendo os requisitos funcionais.

Corolário 7 (Projeto desacoplado com menos informação) Procure um projeto desacoplado que requer menos informação que projetos acoplados em satisfazer um conjunto de requisitos funcionais.

Corolário 8 (Reangularidade efetiva de um escalar) A reangularidade efetiva R para uma matriz de acoplamento escalar ou elemento de matriz é 1.

A.2 Teoremas Gerais de Projeto

Teorema 1 (Acoplamento devido a um número insuficiente de parâmetros de projeto (PPs)) Quando o número de parâmetros de projeto (PPs) é menor que o número de requisitos funcionais (RFs), ou ele resulta em um projeto acoplado ou os requisitos funcionais (RFs) não poderão ser satisfeitos.

Teorema 2 (Redução do acoplamento de um projeto acoplado) Quando um projeto é acoplado devido a um número de requisitos funcionais (RFs) maior que o número de parâmetros de projeto (PPs) ($m > n$), seu acoplamento pode ser reduzido pela adição de novos parâmetros de projeto (PPs) de forma a tornar o número de requisitos funcionais (RFs) igual ao número de parâmetros de projeto (PPs) se um subconjunto da matriz de projeto contendo $n \times n$ elementos constitui uma matriz triangular.

Teorema 3 (Projeto redundante) Quando existem mais parâmetros de projeto (PPs) que requisitos funcionais (RFs), o projeto ou é um projeto redundante ou é um projeto acoplado.

Teorema 4 (Projeto ideal) Em um projeto ideal, o número de parâmetros de projeto (PPs) é igual ao número de requisitos funcionais (RFs) e os requisitos funcionais são sempre mantidos independentes uns dos outros.

Teorema 5 (Necessidade de um novo projeto) Quando um conjunto de requisitos funcionais (RFs) é alterado pela adição de um novo requisito funcional (RF), pela substituição de um dos requisitos funcionais (RFs) por um novo ou pela seleção de um conjunto completamente diferente de requisitos funcionais (RFs), a solução do projeto dada pelos parâmetros de projeto (PPs) originais não pode satisfazer o novo conjunto de requisitos funcionais (RFs). Consequentemente, uma nova solução de projeto deve ser buscada.

Teorema 6 (Independência de caminho do projeto desacoplado) O conteúdo de informação de um projeto desacoplado é independente da sequência pela qual os parâmetros de projeto (PPs) são mudados para satisfazer um dado conjunto de requisitos funcionais (RFs).

Teorema 7 (Dependência de caminho de projeto acoplados e semiacoplados) O conteúdo de informação de um projeto acoplado ou semiacoplado depende da sequência pela qual os parâmetros de projeto (PPs) são mudados e dos caminhos específicos de mudança destes parâmetros de projeto (PPs).

Teorema 8 (Independência e a faixa de variação de projeto) Um projeto é um projeto desacoplado quando a faixa de variação especificada pelo projetista é maior que

$$\left(\sum_{i \neq j, j=1}^n \frac{\partial FR_i}{\partial DP_j} \Delta DP_j \right)$$

Neste caso, os elementos fora da diagonal da matriz de projeto podem ser desconsiderados do projeto.

Teorema 9 (Projeto para “manufaturabilidade”) Para um produto ser manufaturável com confiabilidade e robustez, a matriz de projeto para o produto [A], (que relaciona o vetor de requisitos funcionais (RFs) para o produto com o vetor dos parâmetros de projeto (PPs) do produto), multiplicada pela matriz de projeto do processo de manufatura [B] (que relaciona o vetor de requisitos funcionais (RFs) para o processo de manufatura com o vetor dos parâmetros de projeto (PPs) do processo), deve resultar ou em uma matriz diagonal ou em uma matriz triangular. Consequentemente, quando tanto [A] quanto [B] representam um projeto acoplado, a independência dos requisitos funcionais (RFs) e um projeto robusto não podem ser alcançados. Quando as matrizes forem matrizes triangulares completas, ambas devem ser triangulares superiores ou ambas devem ser triangulares inferiores para que o processo de manufatura satisfaça a independência dos requisitos funcionais (RFs).

Teorema 10 (Modularidade das medidas de independência) Supondo que uma matriz de projeto [DM] possa ser particionada em submatrizes quadradas que somente possuam elementos diferentes de zero na diagonal principal, então, a reangularidade e a semangularidade para [DM] são iguais ao produto das suas medidas correspondentes para cada submatriz.

Teorema 11 (Invariância) A reangularidade e a semangularidade para uma matriz de projeto [DM] não varia com a mudança da ordem dos requisitos funcionais (RFs) e dos parâmetros de projeto (PPs) enquanto forem preservadas as associações entre cada requisito funcional (RF) e seus correspondentes parâmetros de projeto (PPs).

Teorema 12 (Soma da informação) A soma da informação para um conjunto de eventos também é informação, desde que probabilidades condicionais apropriadas sejam usadas quando os eventos não forem estatisticamente independentes.

Teorema 13 (Conteúdo de informação de todo o sistema) Se cada parâmetro de projeto (PP) for probabilisticamente independente dos outros parâmetros de projeto (PPs), o conteúdo de informação total do sistema é a soma da informação de todos os eventos individuais associados com o conjunto de requisitos funcionais (RFs) que devem ser satisfeitos.

Teorema 14 (Conteúdo de informação de projetos acoplados versus desacoplados) Quando o estado dos requisitos funcionais (RFs) é mudado de um estado para o outro no domínio funcional, a informação requerida para a mudança é maior para projetos acoplados que para projetos desacoplados.

Teorema 15 (Interface projeto-manufatura) Quando um sistema de manufatura compromete a independência dos requisitos funcionais (RFs) do produto, ou o projeto do produto deve ser modificado ou um novo processo de manufatura deve ser projetado e/ou utilizado para manter a independência dos requisitos funcionais (RFs) do produto.

Teorema 16 (Igualdade do conteúdo de informação) Todos os conteúdos de informação que são relevantes para a tarefa de projetar são igualmente importantes,

não importando sua origem física, e nenhum fator de ponderação deverá ser aplicado a eles.

Teorema 17 (Projeto na ausência de informação completa) O projeto pode ser feito mesmo na ausência de informação completa apenas no caso de um projeto semiacoplado se a informação faltante está relacionada com elementos fora da diagonal.

Teorema 18 (Existência de um projeto desacoplado ou semiacoplado) Sempre irá existir um projeto desacoplado ou semiacoplado que possua menor conteúdo de informação que um projeto acoplado.

Teorema 19 (Robustez do projeto) Um projeto desacoplado e um projeto semiacoplado são mais robustos que um projeto acoplado no sentido de que é mais fácil reduzir o conteúdo de informação de projetos que satisfazem o axioma da independência.

Teorema 20 (Faixa de variação de projeto e acoplamento) Se as faixas de variação de projeto para projetos desacoplados ou semiacoplados forem limitadas, os projetos podem se tornar projetos acoplados. No sentido contrário, se as faixas de variação de projeto para projetos acoplados forem relaxadas, eles podem se tornar projetos desacoplados ou semiacoplados.

Teorema 21 (Projeto robusto quando o sistema possui uma função de distribuição de probabilidade não uniforme) Se a função de distribuição de probabilidade do requisito funcional (RF) na faixa de variação de projeto não é uniforme, a probabilidade de sucesso é igual a 1 quando a faixa de variação do sistema está dentro da faixa de variação de projeto.

Teorema 22 (Robustez comparativa de um projeto semiacoplado) Dadas as faixas de variação de projeto máximas para um dado conjunto de requisitos funcionais (RFs), projetos semiacoplados não podem ser tão robustos quanto projetos desacoplados em que as tolerâncias permitidas para parâmetros de projeto (PPs) de um projeto semiacoplado são menores que aqueles para um projeto desacoplado.

Teorema 23 (Diminuindo a robustez de um projeto semiacoplado) A tolerância permitida e, portanto, a robustez de um projeto semiacoplado com uma matriz triangular completa diminui com um aumento no número de requisitos funcionais (RF).

Teorema 24 (Agendamento ótimo) Antes que um agendamento para a movimentação de um robô ou agendamento de manufatura possa ser otimizado, o projeto das tarefas deve ser feito de maneira a satisfazer o axioma de independência pela adição de desacopladores para eliminar o acoplamento. Os desacopladores podem ser na forma de um hardware isolado ou buffer.

A.3 Teoremas Relacionados com Projeto de *software*

Teorema *Soft* 1 (Conhecimento requerido para operar um sistema desacoplado) Sistemas de *software* ou *hardware* desacoplados podem ser operados sem um conhecimento preciso sobre elementos do projeto (módulos) se o projeto é verdadeiramente um projeto desacoplado e se as saídas dos requisitos funcionais (RFs) puderem ser monitoradas para permitir um controle dos requisitos funcionais (RFs).

Teorema *Soft* 2 (Tomada de decisões corretas na ausência do conhecimento completo para um projeto semiacoplado com controle) Quando o sistema de *software* é um projeto semiacoplado, os requisitos funcionais (RFs) podem ser satisfeitos pela mudança dos parâmetros de projeto (PPs) se a matriz de projeto é conhecida para a extensão que o conhecimento a respeito da sequencia apropriada é dada, mesmo que não haja um conhecimento preciso a respeito dos elementos do projeto.