UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

RAFAEL GORSKI MORENO SOUZA

PROBLEM-BASED SRS: MÉTODO PARA ESPECIFICAÇÃO DE REQUISITOS DE *SOFTWARE* BASEADO EM PROBLEMAS

DISSERTAÇÃO DE MESTRADO

CURITIBA
2016

RAFAEL GORSKI MORENO SOUZA

PROBLEM-BASED SRS: MÉTODO PARA ESPECIFICAÇÃO DE REQUISITOS DE *SOFTWARE* BASEADO EM PROBLEMAS

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Mestre em Ciências.

Orientador: Prof. Dr. Paulo Cézar Stadzisz

CURITIBA

2016

Dados Internacionais de Catalogação na Publicação

S729p 2016

Souza, Rafael Gorski Moreno
Problem-Based SRS : método para especificação de requisitos de *software* baseado em problemas / Rafael Gorski Moreno Souza .-- 2016.
145 f.: il.; 30 cm.

Disponível também via World Wide Web.
Texto em português, com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Computação Aplicada, Curitiba, 2016.
Bibliografia: f. 126-129.

1. Engenharia de requisitos. 2. Especificações. 3. Software - Desenvolvimento. 4. Serviços ao cliente. 5. Solução de problemas. 6. Controle de qualidade. 7. Teoria axiomática dos conjuntos. 8. Modelagem de negócios. 9. Computação - Dissertações. I. Stadzisz, Paulo Cézar, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. III. Título.

CDD: Ed. 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 45

Aos 23 dias do mês de agosto de 2016 realizou-se na sala C-301 a sessão pública de Defesa da Dissertação de Mestrado intitulada "Problem Based SRS: Método para Especificação de Requisitos de Software Baseado em Problemas", apresentado pelo aluno Rafael Gorski Moreno Souza como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração "Engenharia de Sistemas Computacionais", linha de pesquisa "Engenharia de Software".

Constituição da Banca Examinadora:

Prof. Dr. Paulo Cézar Stadzisz (Presidente) UTFPR – CT _____

Prof. Dr Jean Marcelo Simão UTFPR – CT _____

Prof. Dr. Andrey Ricardo Pimentel UTFPR – CT _____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

RESUMO

SOUZA, Rafael Gorski Moreno. *Problem-Based SRS*: Método para Especificação de Requisitos de *Software* baseado em Problemas. 2016. 145f. Dissertação de Mestrado - Programa de Pós-graduação em Computação Aplicada - PPGCA, Universidade Tecnológica Federal do Paraná. Curitiba, 2016.

Especificação de requisitos é reconhecida como como uma atividade critica nos processos de desenvolvimento de *software* por causa de seu impacto nos riscos do projeto quando mal executada. Um grande conjunto de estudos discute aspectos teóricos, proposições de técnicas e práticas recomendadas para a Engenharia de Requisitos (ER). Para ter sucesso, ER tem que assegurar que requisitos especificados são completos e corretos, o que significa que todas as intenções dos *stakeholders* são cobertas pelos requisitos e que não existem requisitos desnecessários. Entretanto, a captura precisa das intenções *stakeholders* continua sendo um desafio e é o maior fator para falhas em projetos de *software*. Esta dissertação apresenta um novo método denominado "*Problem-Based SRS*" que tem como objetivo melhorar a qualidade da especificação de requisitos de *software* (SRS – *Software Requirements Specification*) no sentido de que os requisitos especificados forneçam uma resposta adequada para os problemas dos clientes reais. Neste método, o conhecimento sobre os requisitos de *software* é construído a partir do conhecimento sobre os problemas do cliente. O *Problem-Based SRS* consiste de uma organização de atividades e resultados através de um processo que contem cinco etapas. O método fornece suporte ao time de engenharia de requisitos para analisar sistematicamente o contexto de negócio e especificar os requisitos de *software*, considerando o vislumbre e a visão do *software*. Os aspectos de qualidade das especificações são avaliados utilizando técnicas de rastreabilidade e princípios do *axiomatic design*. Os casos de estudo realizados e apresentados nesta dissertação apontam que o método proposto pode contribuir de forma significativa para uma melhor especificação de requisitos de *software*.

Palavras-chave: Engenharia de Requisitos, Especificação de Requisitos de *Software*, Problemas do Cliente.

ABSTRACT

SOUZA, Rafael Gorski Moreno. *Problem-Based SRS: Method for Sofware Requirements Specification based on Problems*. 2016. 145f. Dissertação de Mestrado - Programa de Pós-graduação em Computação Aplicada - PPGCA, Universidade Tecnológica Federal do Paraná. Curitiba, 2016.

Requirements specification has long been recognized as critical activity in software development processes because of its impact on project risks when poorly performed. A large amount of studies addresses theoretical aspects, propositions of techniques, and recommended practices for Requirements Engineering (RE). To be successful, RE have to ensure that the specified requirements are complete and correct what means that all intents of the stakeholders in a given business context are covered by the requirements and that no unnecessary requirement was introduced. However, the accurate capture the business intents of the stakeholders remains a challenge and it is a major factor of *software* project failures. This master's dissertation presents a novel method referred to as "*Problem-Based SRS*" aiming at improving the quality of the Software Requirements Specification (SRS) in the sense that the stated requirements provide suitable answers to real customer´s businesses issues. In this approach, the knowledge about the software requirements is constructed from the knowledge about the customer´s problems. *Problem-Based SRS* consists in an organization of activities and outcome objects through a process that contains five main steps. It aims at supporting the software requirements engineering team to systematically analyze the business context and specify the software requirements, taking also into account a first *glance* and vision of the software. The quality aspects of the specifications are evaluated using traceability techniques and *axiomatic design* principles. The cases studies conducted and presented in this document point out that the proposed method can contribute significantly to improve the software requirements specification.

Keywords: *Requirements Engineering, Software Requirements Specification, Customer´s Problems.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS, ACRÔNIMOS E ABREVIATURAS

Sigla	Original	Tradução
AD	*Axiomatic Design*	Projeto Axiomático
AERBPA	*Requirements specification approach based on Axiomatic Design*	*Abordagem de Especificação de Requisitos Baseado em Projeto Axiomático*
C	*Constraint*	Restrição
CI	*Informational Content*	*Conteúdo da Informação*
CN	*Customer Need*	Necessidade do Cliente
CP	*Customer Problem*	Problema do Cliente
CRM	*Customer Relationship Management*	*Gestão de Relacionamento com o Cliente*
DP	*Design Parameter*	Parâmetro de Projeto
e.g.	*Exempli gratia*	*por exemplo*
ER	*Requirements Engineering*	Engenharia de Requisitos
FR	*Functional Requirement*	Requisito Funcional
GORE	*Goal Oriented Requirements Engineering*	Engenharia de Requisitos Orientada a Objetivos
i.e.	*Id est*	*isto é, ou seja*
IHM	*Human Machine Interface*	*Interface Homem Máquina*
LAN	*Local Area Network*	*Interface Local de Rede*
NFR	*Non-Functional Requirement*	Requisito Não-Funcional
NI	*National Instruments*	*National Instruments*
OPD	*Object Process Diagram*	Diagrama Processo Objeto
OPL	*Object Process Language*	Linguagem Processo Objeto
OPM	*Object Process Methodology*	Metodologia Processo Objeto
PDF	*Probability density function*	*Função densidade de probabilidade*
PV	*Project Variables*	Variaveis de Projeto
RTM	*Requirements Traceability Matrix*	*Matriz de rastreabilidade de requisitos*
RUP	*Rational Unified Process*	Processo Unificado
SE	*Software Engineering*	Engenharia de *Software*

SEBOK	*Guide to System Engineering Body of Knowledge*	Guia para Corpo de Conhecimento Engenharia de Sistemas
SG	*Software Glance*	Vislumbre de *Software*
SRS	*Software Requirements Specification*	Especificação de Requisitos de *Software*
SV	*Software Vision*	Visão de *Software*
SWEBOK	*Guide to Software Engineering Body of Knowledge*	Guia para Corpo de Conhecimento Engenharia de *Software*
UML	*Unified Modeling Language*	Linguagem Modelagem Unificada
UP	*Unified Process*	Processo Unificado
v.g.	*Verbi gratia*	*por exemplo*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem como objetivo apresentar a visão geral da dissertação descrevendo o tema do trabalho, seu contexto e os principais problemas que motivaram sua realização. São apresentados, também, os objetivos desta dissertação e o método empregado. Este capítulo contém as seguintes subseções: Subseção 1.1 apresenta o tema, o contexto e a motivação da dissertação. Subseção 1.2 apresenta os objetivos dela. A subseção 1.3 apresenta o método utilizado. Finalizando, a subseção 1.4 apresenta a estrutura de organização do trabalho.

1.1 MOTIVAÇÃO

Software é uma parte importante dos sistemas modernos e é empregado amplamente em todas áreas da economia. Países utilizam *software* para controlar recursos críticos incluindo energia, defesa, segurança e água. Indústrias fazem uso de computadores e *software*s para modelar e supervisionar seus processos. A indústria de telecomunicações constantemente cria novos serviços baseados em *software*. A indústria de entretenimento (*e.g.,* jogos e filmes) faz uso intenso de *software* e atinge milhões de usuários. Igualmente, outras áreas também beneficiam-se de novos sistemas de *software* alavancando o mercado para um crescimento contínuo.

A crescente complexidade para produzir *software* e o aumento constante da demanda do mercado exigem abordagens profissionais para especificar e construir sistemas de *software*. A Engenharia de *Software* (ES) é o campo de conhecimento para atender esta exigência, fornecendo ferramentas e métodos sistemáticos, disciplinados e quantificáveis para o desenvolvimento, operação e manutenção de *software* (BOURQUE e FAIRLEY, 2014). A Engenharia de *Software* auxilia desenvolvedores a aplicar o conhecimento cientifico e técnico em todos os aspectos que ab*range*m a produção de *software*. O Guia *Software Engineering Body Of Knowledge* (SWEBOK) da *Institute of Electrical and Electronics Engineers* (IEEE) fornece uma visão consistente sobre SE e é um importante guia para ser considerado em qualquer fase do ciclo de vida de um *software* (BOURQUE e FAIRLEY, 2014).

A ER inicia o ciclo de vida de *software* e define as intenções e restrições para um *software*. O SWEBOK coloca ER como sua primeira área de conhecimento de ES. ER explora e aplica técnicas de elicitação, análise e documentação de requisitos de *software*. Existem numerosas fontes para requisitos de *software* como os negócios, o conhecimento de domínio, o ponto-de-vista dos *stakeholders* (*i.e.,* aqueles que possuem algum interesse no sistema a ser desenvolvido ou que participam no processo de desenvolvimento do sistema), os processos de negócio e os de aspectos ambientais. O analista de *software* necessita verificar as fontes de requisitos e garantir que as questões mais importantes do negócio são abordadas.

A atividade de elicitação permite que o analista entenda os problemas que um *software* deve resolver. Existem diversas técnicas de elicitação de requisitos, elas podem variar de tradicionais entrevistas até abordagens de estórias de usuários. Independentemente de qual abordagem de elicitação é utilizada, o analista deve assegurar a consistência da informação. A documentação de requisitos, comumente referenciada como Documento de Especificação de Requisitos *Software* (SRS) desempenha um papel importante em estabelecer a base para o acordo entre as partes sobre o que o *software* destina-se a fazer e o que é esperado que seja feito (BOURQUE e FAIRLEY, 2014).

A Engenharia de Requisitos de *software* é reconhecida como uma área crítica na engenharia de *software* porque impacta nos riscos do projeto quando mal executada. Em muitos casos, existe uma lacuna entre o *software* entregue e o esperado pelos *stakeholders*. “A precisão na captura de requisitos de um sistema é a maior fator de falha, com 90% dos grandes projetos de *software*” (DAVIS, *et al.*, 2006). Uma parte significativa das dificuldades para especificação de requisitos de *software* é devida à complexidade em escrever com precisão as sentenças que detalham os requisitos assegurando “assim” as características de uma boa SRS (IEEE, 1998; ISO, 2011). Entretanto, as principais dificuldades ao especificar requisitos de *software* vêm de uma questão muito mais complexa. Os requisitos devem capturar com precisão as intenções de negócio dos *stakeholders*. Desta forma, o analista de *software* deve entender os conceitos, dependências, objetivos de negócio e os processos dos domínios dos

stakeholders de forma que seja possível escrever requisitos adequados. Este entendimento geralmente não é fácil de alcançar por causa da distância entre o domínio técnico no qual o *software* é especificado e o do negócio ou domínio do cliente no qual os problemas emergem.

Pesquisas sobre conceitos e métodos para aprimorar a especificação de requisitos de *software* têm sido desenvolvidas pelo Grupo de Pesquisa em Engenharia de *Software* da UTFPR, no campus de Curitiba, sob a coordenação do professor Dr. Paulo Cézar Stadzisz. Entre outros aspectos, tem-se investigado como o conhecimento é criado e organizado no domínio do cliente e como este conhecimento conduz, por relação causal, à especificação de requisitos de *software* consistentes e coerentes com as demandas dos clientes ou mercado.

Inicialmente, delineou-se uma abordagem para especificação de requisitos considerando três domínios: Domínio dos Problemas, Domínio das Necessidades e Domínio Funcional. Sobre estes domínios, definiu-se técnicas para mapear os conceitos de um domínio em conceitos dos outros domínios de forma a estabelecer a relação causal existente entre eles (PEREIRA, 2011) e permitir ao analista um melhor entendimento das relações entre os conceitos que conduz à especificação dos requisitos. As técnicas estudadas envolvem a teoria de Projeto Axiomático (ou do inglês *Axiomatic Design*) cuja intenção é fornecer meios consistentes para avaliar a qualidade das especificações, em analogia ao que esta teoria propõe para a atividade de *design* (SUH, 1990). O *Axiomatic Design* (AD) foi criado por Nam P. Suh em 1978 como uma teoria e metodologia para projetos (do inglês, *design*) de engenharia e que é fundamentada em dois axiomas: o axioma da independência funcional (ou Axioma 1) e o axioma do conteúdo da informação (ou Axioma 2) (SUH, 1990). O AD foi desenvolvido com o objetivo de aumentar o sucesso em projetos de engenharia (*e.g.,* mecânica, *software*, *hardware*, sistemas, materiais e sistemas de manufatura) utilizando elementos de *design* e processo, como os conceitos de axiomas, domínios, *zigzagging* e matrizes de projeto.

Dentro do contexto exposto, e considerando o estágio atual das pesquisas no grupo de ES na UTFPR campus Curitiba, os problemas de interesse deste estudo de mestrado são relacionados a seguir.

Problema 1 - Compreensão do Domínio do Cliente

O “domínio do cliente” (SUH, 2001) representa o conjunto dos conhecimentos (*i.e.,* dos atributos) relacionados às demandas de um cliente ou mercado sobre um produto ou serviço a ser entregue ou desenvolvido. Trata-se, ainda, de um domínio pouco explorado quando se considera os processos de desenvolvimento de sistemas em geral (Harutunian, 1996; Tate, 1999; Pereira, 2011; Thompson, 2013). Alguns poucos trabalhos de pesquisa, como Pereira (2011) e Thompson (2013), elaboraram discussões sobre como este domínio pode ser representado. Entretanto, o conhecimento presente no domínio do cliente ainda é vago ou difícil de definir.

Problemas relacionados à não compreensão do domínio do cliente são citados no estudo de Thompson (2013) que aponta uma “confusão” na interpretação dos conceitos de requisitos, parâmetros de projeto e restrições. Isto sugere que os engenheiros não dispõem de métodos ou ferramentas para manipular adequadamente informações fora dos domínios funcional, de projeto e de processo.

Este problema da compreensão do domínio do cliente leva à seguinte pergunta de pesquisa: “Qual é o conhecimento essencial do domínio do cliente e como este conhecimento pode ser estruturado em torno dos domínios do problema e das necessidades visando orientar o projetista na análise de requisitos de um *software*? ”.

Problema 2 - Compreensão sobre Relacionamentos entre Domínios

A compreensão sobre o relacionamento dos domínios é tratada por dois pontos de vista neste trabalho (*i*) a análise dos resultados da especificação de requisitos de *software* através dos processos de mapeamento e *zigzagging* dos problemas, necessidades e requisitos e (*ii*) a rastreabilidade de problemas, necessidades e requisitos ao longo do processo de *design*.

Os domínios demarcam as diferentes atividades de *design* e ab*range*m a interação entre “o que se espera alcançar” e “como escolhemos satisfazer esta necessidade”. Esta interação construída através de um processo que envolve o relacionamento entre domínios pode conter soluções que são consideradas não ideais. A teoria do AD fornece aparatos para verificar se uma solução (*design*) é

satisfatória. Os domínios são compostos de informações hierárquicas por meio de matrizes de projeto. As matrizes de projeto (*Design Matrix* ou DM) são utilizadas para relacionar domínios e isto é feito através do processo de *zigzagging* (SUH, 2005).

O Guia SWEBOK enfatiza que a rastreabilidade fornece garantias para os *stakeholders* de que todos os requisitos foram cobertos (BOURQUE e FAIRLEY, 2014). A IEEE 29148 (2011) aponta que a rastreabilidade de requisitos se preocupa com a identificação da origem dos requisitos e o prognóstico dos efeitos dos requisitos ao longo do projeto de *software*. A rastreabilidade é fundamental para três tipos de análises: a de cobertura de realização, a de conformidade e a de impacto. A aplicação das análises de rastreabilidade nos domínios de problemas e necessidades pode beneficiar diretamente o projeto de *software*.

Este problema da compreensão sobre o relacionamento entre domínios leva à seguinte pergunta de pesquisa: “Qual é a maneira como as informações são relacionadas entre os domínios de problema e necessidade e como é realizado o mapeamento, *zigzagging* e rastreabilidade visando garantir a qualidade das especificações produzidas? ”.

Problema 3 - Processo de Especificação de Requisitos

Um processo de *software* é um conjunto de atividades que utilizam recursos como entrada para produzir resultados como saída (ISO, IEC e IEEE, 2010). O processo de especificação de requisitos utiliza um conjunto de atividades inter-relacionadas que transformam os recursos (*i.e.,* problemas e necessidades) no documento de especificação de requisitos de *software* (*e.g.,* SRS). Estas atividades predeterminadas de um processo têm o propósito de fornecer um guia para a execução do processo de especificação de requisitos de *software* de forma que garanta a qualidade dos resultados.

Este problema do processo de especificação de requisitos leva à seguinte pergunta de pesquisa: “Como é construído o processo de especificação de requisitos de *software* levando em consideração os domínios de problema, necessidades e requisitos visando a criação de uma especificação de requisitos de *software* de qualidade?”.

1.2 OBJETIVOS

O objetivo geral desta dissertação de mestrado é desenvolver um método para especificação de requisitos de *software* baseado em problemas, visando melhorar a qualidade das especificações em termos da adequação dos requisitos às intenções do cliente. O método proposto foi denominado "*Problem-Based SRS*".

A partir do objetivo geral, determinou-se os seguintes objetivos específicos:

i. Estabelecer uma base conceitual para o método proposto

O método proposto estará fundamentado em um conjunto de conceitos chave que fornecem a interpretação necessária para o seu entendimento e dos elementos tratados e, ainda, para a sua aplicação. Estes conceitos e inter-relacionamentos serão estabelecidos como base conceitual do método.

ii. Elaborar o método *Problem-Based SRS*

O método será composto por conceitos, processos e resultados. Os processos são compostos por regras básicas e atividades que produzem a especificação de requisitos de *software*. O problema tem papel fundamental no método sendo sua existência o gatilho para o início do desenvolvimento de *software*.

iii. Definir uma nomenclatura padrão para a especificação

A especificação dentro de cada domínio irá empregar uma nomenclatura que orienta o analista na escrita das especificações. Serão propostas nomenclaturas para cada domínio de forma a estabelecer um padrão de especificação.

iv. Propor uma técnica de mapeamento e análise entre domínios

A técnica de mapeamento irá fornecer um guia de como o analista deve mapear as informações dentro e entre os domínios de problemas, necessidades e requisitos. Com este mapeamento realizado, o analista poderá analisar as relações entre problemas, necessidades e requisitos como cobertura e completude relacionado a rastreabilidade.

v. Realizar experimentos para verificação do método

Os experimentos serão utilizados como referencial para o analista sobre como aplicar o método proposto. Estes experimentos são conduzidos com o intuito de criar o conhecimento prático necessário. Serão realizados dois experimentos, um idealizado e outro baseado em um caso real de uma empresa de *software*.

1.3 MÉTODO DE PESQUISA

Esta pesquisa de mestrado pode ser caracterizada por quatro perspectivas distintas: natureza, abordagem do problema, objetivos almejados e aos procedimentos adotados (GIL, 2010).

Na perspectiva da natureza, a pesquisa é classificada como **básica**, porque tem o motivo de obtenção de novos conhecimentos com intuito de buscar a solução de problemas práticos, especificamente o problema de especificação de requisitos de *software*. Na perspectiva da abordagem do problema, a pesquisa é de caráter **qualitativo**, pois envolveu a análise de dados, pelo pesquisador, de forma indutiva. Na perspectiva de objetivos almejados, a pesquisa é classificada como **exploratória** por buscar maior familiaridade com o problema, de modo a torná-lo explícito e possibilitar a formulação de hipóteses, e por ter como propósito a caracterização do problema, com o intuito de identificar as prováveis relações entre as variáveis (MATTAR, 2014).

Quanto aos procedimentos, a pesquisa apoia-se, inicialmente, em uma pesquisa bibliográfica, mapeamento sistemático e análise de trabalhos correlatos. Estes procedimentos subsidiaram a elaboração da proposta teórica.

As etapas do método de pesquisa acompanham o modelo do programa de mestrado do PPGCA, conforme ilustrado na Figura 1. Na primeira etapa foi realizada a revisão bibliográfica com objetivo de construir o conhecimento teórico sobre Engenharia de Requisitos, Especificação de Sistemas e *Axiomatic Design* visando formar a fundamentação para esta pesquisa. Nesta fase foram conduzidos estudos específicos sobre o domínio do cliente (inclusos na Engenharia de Requisitos) e, também, sobre especificação de problemas e necessidades de clientes. Estudou-se, também, os trabalhos anteriores do grupo

de pesquisa sobre *Axiomatic Design*. Os resultados das etapas foram avaliados como requisito de qualificação para o programa de mestrado.

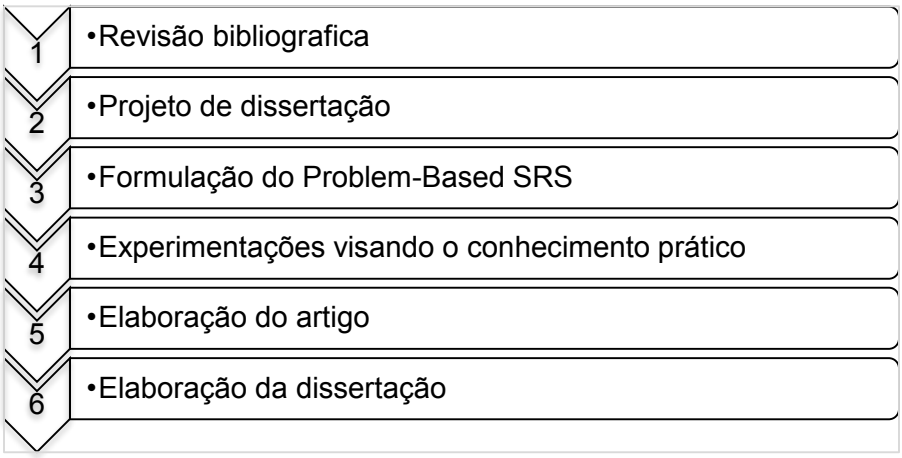


Figura 1 - Etapas da Metodologia do Trabalho de Pesquisa
Fonte: Autoria própria.

A segunda etapa envolveu o projeto de dissertação contendo o tema da pesquisa, seu objetivo geral e objetivos específicos. Nesta etapa procurou-se um alinhamento da pesquisa com os trabalhos anteriores do grupo e com o estado da arte sobre o tema, além da agregação das novas contribuições específicas da pesquisa.

Na terceira etapa foi idealizado o método *Problem-Based SRS* que envolve o estabelecimento da base conceitual, um modelo de processo, nomenclaturas de especificação e técnicas de mapeamento entre domínios e análise do mapeamento.

Na quarta etapa foram aplicados experimentos utilizando o método proposto visando a verificação do método e criação de conhecimento prático. O objetivo foi verificar se o método era viável e trazia benefícios para a especificação dos requisitos de um *software*. Adicionalmente, outras hipóteses foram geradas, orientando assim trabalhos futuros na mesma linha de pesquisa.

Na quinta etapa foi elaborado o artigo chamado "*Problem-Based SRS*" que apresenta o método proposto e resultados parciais da pesquisa dado o momento em que foi elaborado e submetido. O artigo foi enviado e aceito para a revista RESI (Revista Eletrônica de Sistemas de Informação) como requisito para o programa de mestrado.

A etapa final foi composta pela elaboração do documento de dissertação contendo os resultados e conclusões sobre o trabalho de pesquisa.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Neste presente capítulo foi apresentado o contexto no qual este trabalho se insere, a justificativa e os objetivos geral e específicos, além do método de pesquisa.

O Capítulo 2 apresenta a fundamentação teórica que subsidiou o desenvolvimento desta dissertação que envolveu: Engenharia de Requisitos de *Software*, Padrões de Engenharia de *Software* para Requisitos, Abordagens De Especificação de Requisitos e Projeto Axiomático.

O Capítulo 3 apresenta a abordagem desenvolvida para a especificação de requisitos de *software* baseados em problemas, que envolve uma proposta de conceitos e o processo.

No Capítulo 4 é exposto um caso de estudo da aplicação do método *Problem-Based SRS* em um exemplo, que visa mostrar a aplicabilidade do método.

O Capítulo 5 apresenta uma análise do método *Problem-Based SRS*, conduzida em um experimento com o intuito de analisar o método sob o ponto de vista real. Essa análise foi realizada, a partir do projeto do Sistema MicroER que está sendo desenvolvido pelo LIT (Laboratório de Inovação e Tecnologia em Sistemas Embarcados) da UTFPR - campus Curitiba.

No Capítulo 6 são apresentadas as considerações finais, destacando as conclusões, a relevância deste estudo, as contribuições, dificuldades e limitações, bem como oportunidades de trabalhos futuros identificadas.

Por fim, os apêndices contidos neste documento apresentam:

- Apêndice A: Corolários e Teoremas em português do AD.
- Apêndice B: Método utilizado na pesquisa bibliográfica.
- Apêndice C: Notação *Object Process Diagram* (OPD).

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos relevantes que subsidiaram o desenvolvimento desta dissertação que englobam três áreas principais: a engenharia de *software* com a engenharia de requisitos, as abordagens de especificação de requisitos de *software* e a abordagens de especificação de requisitos com *axiomatic design*.

2.1 ENGENHARIA DE *SOFTWARE* E ENGENHARIA DE REQUISITOS

Engenharia de *Software* é a aplicação sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de *software*, isto é, a aplicação de engenharia ao *software*. O SWEBOK é o Guia para o Corpo de Conhecimento de Engenharia de *Software* mantido pela IEEE e que apresenta e promove uma visão consistente sobre engenharia de *software* mundialmente. O SWEBOK é dividido em áreas de conhecimento e especifica o escopo e esclarece o lugar da engenharia de *software* com respeito a outras engenharias. A primeira área de conhecimento citada no SWEBOK é a de Requisitos de *Software* que é composta pelos seguintes tópicos: fundamentos de requisitos de *software*, processo de requisitos, elicitação de requisitos, análise de requisitos, especificação de requisitos, validação de requisitos, considerações práticas e ferramentas de requisitos de *software* (BOURQUE e FAIRLEY, 2014).

O sucesso de um projeto de *software* está ligado a execução com sucesso da engenharia de requisitos (ER). A área de requisitos de *software* requer uma quantidade significativa de recursos que é, em média, de 28 por cento dos recursos totais (HOFMANN e LEHNER, 2001) mostrando, assim, a importância da área de requisitos em um projeto de *software*. Adicionalmente, de acordo com Davey e Parker (2015), existe um entendimento comum de que a correção de erros em uma elicitação de requisitos deficiente é mais cara que a correção de outros erros durante o projeto de *software* (DAVEY e PARKER, 2015). Zowghi e Coulin (2005) esclarecem que a atividade de elicitação de requisitos de *software* é o processo de busca, revelação, aquisição e elaboração de requisitos de sistemas de *software*. O processo de elicitação preocupa-se com a aprendizagem e entendimento das necessidades dos clientes e tem objetivo de comunicar as necessidades para os desenvolvedores de *software*

(ZOWGHI e COULIN, 2005). A elicitação de requisitos permanece, essencialmente, uma tarefa centrada no ser humano que utiliza como principal princípio a comunicação efetiva entre os principais *stakeholders* (BOURQUE e FAIRLEY, 2014).

Davey e Parker (2015) classificam nove categorias de problemas que afetam ER, conforme listado a seguir.

i) Em ER existem aspectos humanos que impedem a comunicação simples entre o analista e o cliente.
ii) A linguagem humana nem sempre é adequada para expressar soluções tecnológicas.
iii) Requisitos mudam ao longo do projeto.
iv) Clientes pedem (as vezes) requisitos que as organizações não necessitam.
v) Clientes não sabem dizer quais são as necessidades de negócio.
vi) Alguns clientes não querem ajudar o projeto.
vii) Sintomas que não são problemas são geralmente reportados.
viii) ER nem sempre é determinístico.

Dentre as nove categorias de Davey e Parker (2015), as categorias iv e v tratam de necessidades. A categoria de problema *iv* claramente representa o mal entendimento sobre as dependências entre os requisitos e as reais necessidades dos clientes. Similarmente, a categoria *v* representa o mal entendimento sobre as dependências de necessidades e problemas dos clientes. A falta de sucesso no reconhecimento dos Problemas dos Clientes (CPs) e as Necessidades dos Clientes (CNs) como elementos distintos na especificação de requisitos sugerem oportunidades de extensão nos estudos atuais sobre o assunto.

2.1.1 Padrões da IEEE para Requisitos de *Software*

Nesta subseção serão abordados os padrões recomendados pelo Guia SWEBOK que são relevantes para o trabalho. Em especial, os padrões IEEE 29148 (2011) e IEEE 24765 (2010) auxiliam no entendimento da execução de processos de engenharia de requisitos. Os padrões têm como objetivo fornecer uma definição normativa e recomendações sobre o formato dos itens de

informação e documentação que resultam da implementação dos processos. Os parágrafos a seguir apresentam brevemente tópicos de interesse para este estudo, que são o vocabulário para ER, a especificação de requisitos e a rastreabilidade de requisitos.

Vocabulário e Termos

A utilização de termos comuns auxilia a comunicação entre profissionais e equipes no entendimento que envolve o processo de engenharia de requisitos de *software*. O padrão internacional "IEEE 24765: Vocabulário para Engenharia de Sistemas e *Software"* apresenta uma terminologia padrão para a área de engenharia de sistemas e *software*. Como preparação para a escrita de requisitos de *software*, a utilização de termos universais ajuda no entendimento e discussões futuras dentro dos processos de requisitos. As seguintes definições foram selecionadas e adaptadas do padrão IEEE 24765 (2010) a este trabalho (ISO, IEC e IEEE, 2010):

- **Cliente**: do inglês *customer* é a organização, empresa ou pessoa que recebe um produto ou serviço.
- **Engenharia de Requisitos**: ciência e disciplina preocupada em analisar e documentar requisitos.
- **Especificação de Requisitos**: documento que especifica os requisitos de um sistema ou componente. Tipicamente inclui requisitos funcionais, requisitos de desempenho, requisitos de interface, requisitos de projeto e padrões de projeto.
- **Matriz de Rastreabilidade de Requisitos (RTM)**: tabela que liga requisitos com suas origens e os rastreia ao longo do ciclo de vida do projeto.
- **Particionamento de Requisitos**: separação ou decomposição de um requisito de alto nível do projeto em sucessivos requisitos detalhados de mais baixo nível.
- **Rastreabilidade de Requisitos:** identificação e documentação do caminho de derivação (para cima) e alocação (para baixo) dos requisitos dentro da hierarquia de decomposição dos requisitos. Associação perceptível entre um requisito e requisitos relacionados, implementações e verificações.
- ***Requisito***: condição ou capacidade necessitada por um usuário para resolver (do inglês "*to solve"*) um problema ou alcançar um objetivo.

- **Restrição**: limitação ou requisito implícito que restringe o projeto da solução e que não pode ser modificado pelo desenvolvedor.

Adicionalmente, o SWEBOK descreve que um requisito é uma propriedade que deve ser apresentada por algo que resolva algum problema no mundo real (*e.g.,* automatizar uma tarefa que apoia os processos de negócio de uma empresa e controlar um dispositivo) (BOURQUE e FAIRLEY, 2014). A definição dos termos básicos enseja a revisão sobre especificação de requisitos que o padrão IEEE 29148 (2011) recomenda. O próximo parágrafo esclarece as principais orientações relacionadas à escrita dos requisitos funcionais.

Especificação de Requisitos

O padrão internacional "IEEE 29148 - Engenharia de Requisitos para Ciclo de Vida de Processos de Sistema e *Software*" (2011) foi escolhido para este trabalho por trazer a forma mais atual vista pela IEEE sobre a escrita de requisitos e grupos de requisitos. O padrão indica que a construção dos requisitos deve ser bem executada e deve garantir que o requisito é verificável e que resolve um problema ou alcança um objetivo de um *stakeholder*. Um bom requisito deve ser qualificado de condições e medidas e limitado com restrições. Para a construção dos requisitos, o padrão recomenda atenção aos seguintes pontos:

- Requisitos constituem disposições obrigatórias e usam "deve" na sua forma escrita;
- Sentenças sobre fatos, questões futuras, propósitos não obrigatórios ou disposições não obrigatórias usam o termo "desejo". Porém, a utilização do termo "desejo" pode trazer interpretações indesejadas por tanto recomenda-se a não utilização do termo;
- Preferências, objetivos desejados, não obrigatórios ou disposições não obrigatórias devem usar o termo "deveria";
- Sugestões, permissões ou disposições não obrigatórias usam o termo "pode";
- Sentenças classificadas como "não requisitos" devem evitar obrigações ou deveres;
- Deve-se utilizar sentenças positivas e evitar sentenças negativas;

- Utilizar voz ativa nas sentenças.

Requisitos podem ser escritos na forma de sentenças em linguagem natural, entretanto estas sentenças podem variar de forma dependendo do profissional que as escreve. Para auxiliar a uniformização da escrita de um requisito, as seguintes propostas de formalização da escrita são recomendadas pela (ISO, IEC e IEEE, 2011).

[Condição] [Sujeito] [Ação] [Objeto] [Restrição]

Exemplo: Quando um sinal x é recebido **[Condição],** o *software* **[Sujeito]** deve configurar **[Ação]** o bit recebido **[Objeto]** a cada dois segundos **[Restrição]**.

[Condição] [Ação ou Restrição] [Valor]

Exemplo: Dentro da área marítima um **[Condição]**, o sistema de radar deve detectar os alvos em uma distância de até **[Ação ou Restrição]** 100 milhas náuticas **[Valor]**.

[Sujeito] [Ação] [Valor]

Exemplo: O *software* de faturas **[Sujeito]** deve apresentar as faturas pendentes **[Ação]** do usuário em ordem decrescente **[Valor]** de cada fatura paga.

A IEEE 29148 (2011) distingue as características dos requisitos em dois grupos: o do requisito individual e de grupos de requisitos. Após a utilização da recomendação de escrita, o profissional de requisitos precisa então verificar se cada requisito é necessário, independente, claro ou não ambíguo, consistente, único, factível, rastreável e verificável. A IEEE 29148 (2011) auxilia com recomendações sobre cada uma destas características para requisitos individuais (ISO, IEC e IEEE, 2011):

- **Necessário**: um requisito define uma capacidade essencial, característica, restringente e/ou um fator de qualidade. Caso este requisito seja removido, uma deficiência será criada e esta não poderá ser substituída por outras capacidades.

- **Independente de implementação**: o requisito deve abordar somente o necessário e suficiente dentro do *software* evitando restrições desnecessárias sobre a implementação física.
- **Não ambíguo**: o requisito é definido de forma que somente tenha uma única interpretação. O requisito é definido de forma simples e fácil de entender.
- **Consistente**: o requisito está livre de conflitos com outros requisitos.
- **Singular**: a sentença que descreve o requisito inclui um único requisito.
- **Factível**: o requisito é tecnicamente viável e não necessita de avanços tecnológicos significativos e cabe dentro das limitações do *software* (e.g. custo, prazo, tecnologia, legal, regulatório) e com um risco aceitável.
- **Rastreável**: cada requisito deve ter um identificador único que facilita a rastreabilidade ao longo do processo de elicitação. É importante que todos os requisitos estejam classificados por ordem de importância, disponibilidade, implementação ou de estabilidade, esta definição é similar a IEEE-830 (1998).
- **Verificável**: cada requisito deve ser claro na forma de como testá-lo e deve ter uma única interpretação, esta definição é similar a IEEE-830 (1998).

Quando requisitos são agrupados, características adicionais necessitam ser levadas em conta como se o grupo de requisitos é completo, consistente, acessível e delimitado.

- **Completo**: um conjunto de requisitos é considerado completo quando não é preciso de mais detalhamento e contém todos os aspectos pertinentes para definição do *software* ou unidade.
- **Consistente**: um conjunto de requisitos que não contém requisitos contraditórios ou duplicados.
- **Accessível**: um conjunto de requisitos que pode ser satisfeito por uma solução factível e obtenível dentro das limitações do projeto (*e.g.,* custo, prazo, tecnologia, legal é regulatório).

- **Delimitado**: um conjunto de requisitos que mantém a identidade de escopo para a solução intencionada sem aumentar o que é necessário para satisfazer as necessidades de um cliente.

A forma de escrita dos requisitos é um ponto fundamental em uma especificação de requisitos de *software*. Todos os requisitos são reunidos em uma especificação de requisitos de *software*. A especificação de requisitos de *software* é realizada dentro de um processo de requisitos de *software*. Para melhor entender quais são as fases de um processo de requisitos de *software* os próximos parágrafos apresentam a visão do SWEBOK.

Rastreabilidade de Requisitos

O Guia SWEBOK enfatiza que é importante, por motivos contratuais, manter a rastreabilidade dos requisitos, pois se a rastreabilidade não é atualizada a cada mudança de requisitos, esta informação pode ser considerada não confiável para análises de impacto. Adicionalmente, enfatiza que a rastreabilidade fornece garantias para os *stakeholders* de que todos os requisitos foram cumpridos (BOURQUE e FAIRLEY, 2014). A IEEE 29148 (2011) aponta que a atividade de rastreabilidade de requisitos é frequentemente utilizada como ponto único de prestação de contas sobre a rastreabilidade do requisito até a fonte e como este requisito foi atendido ao longo do ciclo de vida do *software*.

A rastreabilidade de requisitos se preocupa com a identificação da origem dos requisitos e o prognóstico dos efeitos (*e.g.,* usos, implementações e testes) dos requisitos. A rastreabilidade é fundamental para três tipos de análises: a de cobertura de realização, a de conformidade e a de impacto. Na análise de cobertura de realização verifica-se que todos os requisitos dos *stakeholders* foram cumpridos pelo projeto (*design*) e, também, se cada requisito de baixo nível é justificável. A análise de conformidade é a documentação de que os requisitos dos *stakeholders* foram satisfeitos. A análise de impacto é realizada quando um requisito é alterado. Os requisitos devem ser rastreáveis para cima entendendo o que os motivaram (requisitos de sistema). A rastreabilidade de forma bidirecional deve ser usada para melhorar a integridade e precisão de todos os requisitos habilitando métricas como cobertura, conformidade e complexidade (ISO, IEC e IEEE, 2011).

2.1.2 Processo de Requisitos de *Software*

O Guia SWEBOK esclarece que o processo de requisitos de *software* não é uma atividade realizada a frente do projeto de *software*, mas em vez disto inicia no começo do projeto e continua a ser refinado ao longo do ciclo de vida (BOURQUE e FAIRLEY, 2014). Complementarmente, o Guia SWEBOK coloca como essencial a escolha de um processo de requisitos. Apresenta-se, a seguir duas opções de processos dos autores Sommerville (2010) e Jacobson (1999) que tem visões e abordagens complementares sobre processos de engenharia de requisitos.

Para Sommerville (2010), seu processo de engenharia de requisitos tem a intenção de produzir um documento de requisitos de *software* que especifique um sistema que satisfaça os requisitos dos *stakeholders*. As principais atividades de um processo de engenharia de requisitos podem ser as seguintes (representado na Figura 2) (SOMMERVILLE, 2010):

- **Estudo de Viabilidade**: neste estudo são identificadas e estimadas as necessidades dos usuários que podem ser satisfeitas por um *software*. O estudo considera se o *software* proposto tem um custo aceitável ou restritivo do ponto de vista de negócio. O objetivo é decidir se é possível continuar a análise para um nível mais detalhado. O resultado é a produção de um relatório de viabilidade que indica a estimativa de construção do *software* e a viabilidade de custo baseada no orçamento disponível.
- **Elicitação e Análise**: este é o processo de derivar os requisitos através de técnicas de elicitação como observação, entrevistas, protótipos e *workshops*. Os possíveis *stakeholders* do projeto participam desta fase para ajudar o profissional de requisitos a entender o sistema a ser especificado.
- **Especificação de Requisitos**: esta atividade traduz as informações adquiridas na fase de análise e gera o documento que define um conjunto de requisitos.
- **Validação dos Requisitos**: é a última fase na qual é realizada uma verificação se os requisitos estão realistas, consistentes e completos. Erros são comuns de serem encontrados e devem ser corrigidos.

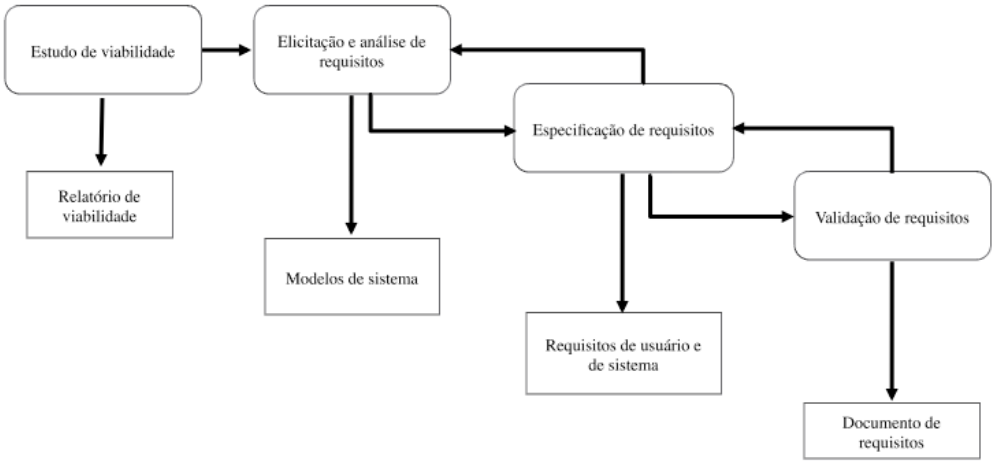


Figura 2 - Processo de Engenharia de Requisitos
Fonte: Sommerville (2010)

Sommerville (2010) lembra que o processo da Figura 2 não necessariamente deve ser seguido simplesmente nesta sequência e que atividades como análise e elicitação continuam durante o ciclo de vida do *software*. Novos requisitos podem surgir de membros do time de desenvolvimento durante o projeto. Abordagens ágeis como *extreme programming* produzem requisitos de forma incremental e podem ser utilizados.

Em uma visão complementar, o Processo Unificado de Desenvolvimento de *Software* (*UP*) provê um processo iterativo baseado em cinco fluxos (*workflows*) que contêm suporte a requisitos de *software* como apresentado na Figura 3 (JACOBSON, BOOCH e RUMBAUGH, 1999). O UP é um processo amplamente conhecido e utilizado para desenvolvimento de *software* e está presente no trabalho de Pimentel (2007) no qual é possível verificar a aplicação do Processo Unificado em detalhe.

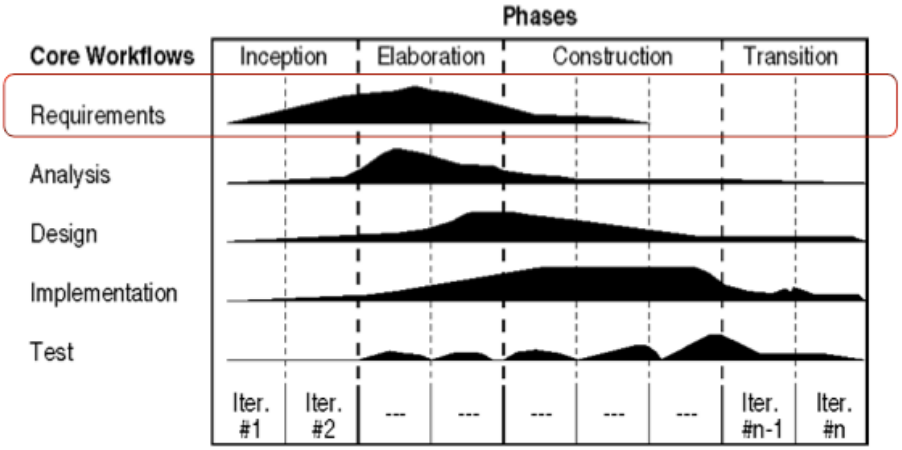


Figura 3 - *Workflow* de Requisitos de *Software* do Processo Unificado
Fonte: Jacobson, Booch e Rumbaugh (1999)

O objetivo do *workflow* de requisitos no UP é desenvolver os modelos de casos de uso ao longo de ciclos incrementais. O modelo de casos de uso pode ser criado ou detalhado durante as iterações como apresentado **destacado** na Figura 3. Os requisitos são desenvolvidos principalmente nas fases de *inception* e *elaboration*. Durante a fase de *inception*, o analista identifica a maioria dos casos de uso para determinar o escopo do projeto. Durante a fase de *elaboration* são definidos os casos de uso restantes e a tarefa de detalhamento dos casos de uso é concluída. Na fase de *construction* podem haver casos de uso em detalhamento, finalização ou modificação. Na fase de *transition*, os casos de uso podem ser modificados em caso de mudança de requisitos (JACOBSON, BOOCH e RUMBAUGH, 1999).

Os processos de engenharia de requisitos aqui apresentados demostram formas de gerenciamento dentro do ciclo de vida de *software*.

2.2 ABORDAGENS DE ESPECIFICAÇÃO DE REQUISITOS

Nesta subseção, são discutidos alguns aspectos da abordagem orientada a objetivos e *problem-frames*. Adicionalmente, como fonte de inspiração, foi analisada a na especificação de problemas e necessidades na interpretação destas abordagens com visões de outros autores na área de modelagem de negócios e engenharia de sistemas.

2.2.1 Abordagem Orientada a Objetivos e Framework I*

Yu (1997) apresenta um modelo e *framework* chamado *i** para capturar as estruturas intencionais do processo do *software* em termos de relacionamento de dependência entre os atores. Neste modelo, Yu argumenta que é necessário saber os “Porquês“ por trás dos “O que“ e “Como“, usualmente mapeados no processo de Engenharia de Requisitos. O *framework i** é composto por um modelo de dependência estratégica e outro de dependência racional e pode ser aplicado para modelagem de negócios.

O modelo de dependência estratégica tem como objetivo mapear o relacionamento de dependência entre os atores no nível da organização e é composto por componentes: atores, objetivos, intenções (*soft goals*), tarefas,

recursos e dependências. O modelo de dependência racional, por sua vez, tem como objetivo mapear os interesses e preocupações dos *stakeholders* que podem ser resolvidos por diversas configurações de sistema ou ambiente.

Na modelagem utilizada por Yu (1997) é apresentado um foco nos atores, interações entre eles e intenções com os componentes do sistema. Em nenhum momento é respondida à pergunta de “porque” o sistema é necessário ou as reais intenções para o desenvolvimento de tal *software*. O autor apresenta que é possível mapear o que os atores “Querem” contra as “Habilidades” do sistema. O modelo apresenta como lidar com os problemas, mas não apresenta como listá-los ou como identificá-los. A premissa é ter os atores identificados e o conjunto de objetivos já decompostos para o início da modelagem de dependências organizacionais e racionais.

Este método apresenta uma abordagem focada exclusivamente nos objetivos estratégicos, deixando o contexto do sistema de lado, comprometendo, assim o processo de decomposição de “subobjetivos” (PEREIRA, 2011). Devido a não utilização do contexto do sistema, ou seus reais problemas e necessidades, o método proposto pode gerar derivações indesejadas e desalinhadas com relação às expectativas do cliente.

A utilização de uma abordagem axiomática como a de Pereira (2011) enriquece o mapeamento entre problemas, necessidades e requisitos funcionais resultando em um *design* com mais chances de sucesso devido ao seu embasamento teórico. Nas subseções seguintes discute-se a utilização de AD com a engenharia de requisitos.

2.2.2 Especificação de Problemas e Necessidades

É de senso comum que os requisitos de *software* têm que satisfazer as intenções dos *stakeholders*. O padrão IEEE utiliza a expressão “da forma que (o requisito) seja aceitável para o cliente” para enfatizar que os requisitos devem atender as intenções dos clientes (ISO, IEC e IEEE, 2010). Assim, alguns autores propõem a validação de um requisito específico através da determinação de sua conformidade com as necessidades do *stakeholder* (HOFMANN e LEHNER, 2001). O termo “necessidade” é comumente empregado para fazer referência à causa ou razão que justifica um requisito de *software* específico

(ZOWGHI e COULIN, 2005). As necessidades poderiam ser a fonte dos requisitos.

Além disso, alguns autores observam que as necessidades dos *stakeholders* colocadas em um produto de *software*, por sua vez, contribuem com a solução de algum problema no mundo real (BOURQUE e FAIRLEY, 2014). De fato, uma solução de *software* desenvolvida de acordo com um conjunto de requisitos específicos deveria resolver problemas relacionados aos clientes (LEFFINGWELL e WIDRIG, 2003).

Dentro das técnicas de especificação de requisitos, a abordagem baseada em objetivos (do inglês, *goal*) (*e.g.,* GORE - *Goal Oriented Requirement Engineering*) tem sido utilizada na engenharia de *software* para modelar requisitos iniciais e não funcionais (GIORGINI, *et al.*, 2002) por meio de objetivos. Objetivos são sentenças prescritivas de intenções a serem satisfeitas por um sistema de *software*. A análise dos objetivos envolve decompor objetivos em subobjetivos por meio de estruturas de “E-” ou “OU-” (VAN LAMSWEERDE, 2004). As implementações comuns da abordagem GORE são KAOS, I*, e NFR (YU, *et al.*, 2011). O método KAOS (*Knowledge Acquisition in autOmated Specification or Keep All Objects Satisfied*) é a mais rica das técnicas de análise de objetivos e possibilita ao analista de sistemas gerar modelos de objetivos para derivação posterior da SRS. No modelo KAOS, as necessidades dos *stakeholders* representam objetivos, responsabilidades, objetos e operações do sistema de interesse (DARDENNE, VAN LAMSWEERDE e FICKAS, 1993).

Com respeito ao entendimento dos problemas do mundo-real, a abordagem de *Problem-Frames* (PF) apresenta uma visão diferente relacionada a problemas, especificações e requisitos de *software*. O “problema” no PF é definido como um problema relacionado ao *software* visto como requisito no contexto do mundo real no qual o *software* tenta ajudar a resolver. PF tem o propósito de satisfazer a necessidade reconhecida (*i.e.,* requisito) através da transformação do mundo físico. A parte do mundo que é transformada, na qual se localizam os requisitos, é chamado de mundo-real. A progressão-do-problema é a atividade principal do PF e cria a especificação baseada nos requisitos (JACKSON, 1999). PF pode ser referenciado como a definição de classes de problema dentro da engenharia de *software* (JACKSON, 2005).

No campo de análise de negócios, o Guia para o Corpo de Conhecimento de Análise de Negócios (BABOK) faz referência a problemas e necessidades usando termos como: *i*) Sentença de Problema descreve problemas em seu estado atual e esclarece qual será a solução de sucesso, *ii*) Necessidade de Negócio é uma sentença com o objetivo de negócio ou um impacto que a solução deve ter no ambiente. De acordo com IIBA (2009), o processo de “definir necessidades de negócio” considera um amplo alcance de alternativas de soluções validando os problemas encontrados (IIBA, 2009).

No contexto de Engenharia de Sistemas, o entendimento profundo das intenções do sistema e como ela são mapeadas em requisitos técnicos do sistema é tão importante quanto na engenharia de *software*, sendo os conceitos apresentados análogos. O Conselho Internacional de Engenharia de Sistemas (INCOSE) emprega os termos “Problema e Oportunidade” para referir-se a assuntos subjacentes às lacunas na estratégia da empresa com respeito aos objetivos e finalidades desejados pela empresa (INCOSE, 2015). A Engenharia de Sistemas também utiliza o termo “necessidades dos *stakeholders*” para representar “necessidades determinadas a partir da comunicação com *stakeholders* internos e externos visando entender suas expectativas, necessidades, requisitos, valores, problemas, assuntos, oportunidades e riscos percebidos”. O termo “necessidades dos *stakeholders*” refere-se a “requisitos de vários *stakeholders* que irão coordenar o projeto, incluindo capacidades funcionais do sistema requerido e/ou serviços; padrões de qualidade; restrições de sistema; restrições de custo; e cronograma. Adicionalmente, o termo “requisitos de sistema” significa “O que o sistema necessita fazer, qual a melhor forma e sob quais condições são requeridos para atender as restrições de projeto e planejamento”.

Especialmente na área corporativa, estudos sobre a modelagem de negócio têm a intenção de entender e modelar funções de negócios, definindo como a empresa cria valor. Quando o modelo de negócio é transmitido para a camada de sistemas da informação, uma certa abstração é necessária para acomodar suas orientações estratégicas. Uma visão unificada para simplificar o modelo de negócio e melhor mapear os valores de negócio e clientes é importante para expressar as necessidades dos clientes. Osterwalder e Pigneur

(2003) propõem uma abordagem chamada "*e-business model*" (OSTERWALDER e PIGNEUR, 2003) e, mais recentemente, apresentada como "*Value Proposition Design*" (VPD) (OSTERWALDER, *et al.*, 2014).

Similarmente, Blank e Dorf (2012) descrevem um estudo sobre problemas dos clientes como o entendimento dos problemas que impactam em uma organização e qual "intensidade de dor" isto pode causar ao cliente. Em outras palavras, como e porque os clientes experimentam os problemas. Esta informação constrói argumentos relevantes para o desenvolvimento de soluções e resolver problemas. Blank e Dorf (2012) sugerem que problemas podem ser expressos como problemas latentes, passivos, ativos ou de visão, dependendo do conhecimento e motivação do cliente para solucioná-lo. De acordo com o VPD, Osterwalder (2014) divide o perfil do cliente em três seções denominadas dores, ganhos e tarefas. As dores são resultados ruins, riscos e obstáculos relacionados às tarefas do cliente. Os ganhos são os resultados esperados pelo cliente e benefícios concretos que se deseja alcançar. Tarefas detalham trabalhos realizados para alcançar os resultados nos negócios diários.

Apesar da grande quantidade de estudos e abordagens propostas para especificar requisitos de *software*, é um senso comum de que a especificação de requisitos continua uma tarefa obscura e desafiadora para as empresas. Termos utilizados na literatura cientifica e na indústria (como apontado nesta seção) não são convergentes criando uma falta de entendimento sobre o assunto. Analistas de *software* podem se sentir confusos quando tentam obter informações de *stakeholders* e outras fontes no negócio ou no domínio de aplicação. Desta forma, a especificação de requisitos de *software* pode ficar incompleta ou incorreta por causa do entendimento inadequado das intenções do projeto.

2.3 ESPECIFICAÇÃO DE REQUISITOS COM *AXIOMATIC DESIGN*

Nesta subseção apresenta-se a introdução e termos da Teoria do *Axiomatic Design* (AD) e sua aplicação na especificação de requisitos de *software*. Nesta revisão são apresentados elementos de *design* e processo juntamente com os conceitos fundamentais como o Axioma da Independência e o Axioma da Informação.

2.3.1 Princípios do *Axiomatic Design*

O *Axiomatic Design* (AD) foi criado por Nam P. Suh em 1978 como uma teoria e metodologia para projetos (do inglês, *design*) de engenharia, sendo ela fundamentada em dois axiomas: o axioma da independência funcional (ou Axioma 1) e o axioma da minimização do conteúdo de informação (ou Axioma 2) (SUH, 1990). O AD foi desenvolvido com o objetivo de aumentar o sucesso em projetos de engenharia (*e.g.,* mecânica, *software*, *hardware*, sistemas, materiais e sistemas de manufatura) utilizando elementos de *design* e processo como os conceitos de domínios, matrizes de projeto e complexidade.

A seguir apresenta-se as definições chave extraídas de Suh (2005) e Pimentel (2007) que contextualizam o AD:

Axioma: axiomas são verdades que não podem ser derivadas mas para as quais não existem contra-exemplos ou exceções. Um axioma não pode ser derivado de outras leis ou princípios da natureza.

Teorema: uma proposição que não é verdade por si só (do inglês *self-evident*), mas para a qual é possível descrever provas com premissas aceitáveis ou axiomas e é estabelecido como lei ou princípio.

Corolários: inferências derivadas de axiomas ou de proposições (teoremas) derivadas de axiomas ou outras proposições comprovadas.

Requisitos Funcionais (FRs): um conjunto independente mínimo de requisitos funcionais que caracteriza completamente as exigências funcionais de um produto (ou *software*, organização ou sistema) dentro do domínio funcional.

Parâmetros de Projeto (DPs): são as principais variáveis físicas dentro do domínio físico que caracterizam o *design* e satisfazem os requisitos funcionais do projeto.

Variáveis de Processo (PVs): são as variáveis chaves no domínio de processo que caracterizam como o processo de produção pode gerar, ou realizar, os parâmetros de projeto especificados (SUH, 2005). No caso de sistemas de *software*, as variáveis de processo podem ser: linhas de código, variáveis locais, compiladores, entre outros (PIMENTEL, 2007).

Restrições (do inglês *constraints*): são limites em soluções aceitáveis. Existem dois tipos de restrições: as restrições de entrada e restrições de sistema. Em engenharia de *software* as restrições são consideradas como requisitos não funcionais (PIMENTEL, 2007).

Axioma da Independência: a solução deve manter a independência funcional dos requisitos do sistema.

Axioma da Informação: a solução deve minimizar o conteúdo de informação de um *design*.

O AD foi criado para ser aplicado a todas as atividades de *design* e inclui componentes que o distinguem de outras propostas de *design*, componentes como os domínios, as hierarquias, o *zigzagging* e axiomas, teoremas e corolários. A relação de cada componente do AD com suas áreas de conhecimento e aplicação estão descritos no Quadro 1. Os elementos de *design* são utilizados como fundamentação para aplicação da metodologia. Os elementos de processo são utilizados para síntese e análise dos elementos de *design*.

Conceito	**Área de Conhecimento**	**Exemplos**
Domínios	Elemento de *Design*	Problemas de *design* podem ser modelados e mapeados entre domínios. Requisitos são articulados em um ambiente neutro de solução.
Hierarquias	Elemento de *Design*	*Design* consiste em uma hierarquia de tarefas de projeto em múltiplos níveis de abstração.
Zigzagging	Elemento de *Design* e Processo	A decisão tomada no nível mais alto da hierarquia do projeto molda e formula problemas de projeto em níveis mais baixos.
Independência	Elemento de *Design* e Processo	Existe uma correlação positiva entre as decisões de manter a independência entre os requisitos funcionais e a melhoria de

		qualidade, assim como a redução de recursos utilizados (*e.g.,* tempo). A independência pode ser modelada em termos de matrizes de projeto.
Conteúdo de Informação	Elemento de *Design* e Processo	Conteúdo de informação significa mensurar a "complexidade" do *design*. A qualidade do *design* necessita ser avaliada e comparada com um conjunto de requisitos claramente definidos.

Quadro 1 - Conceitos e hipóteses do *Axiomatic Design*
Fonte: Tate (1999)

Suh (1990) define projeto, do inglês *design*, como o mapeamento entre requisitos funcionais (FRs) no domínio funcional e os parâmetros de projeto (DPs), no domínio físico. O *design* envolve a interação entre "o que se espera atingir" e "como alcançá-lo", sendo representado pelos domínios ilustrados na Figura 4. Elementos do domínio da esquerda afetam os elementos do domínio da direita e o processo de *design* acontece dentro destes domínios através de mapeamentos.

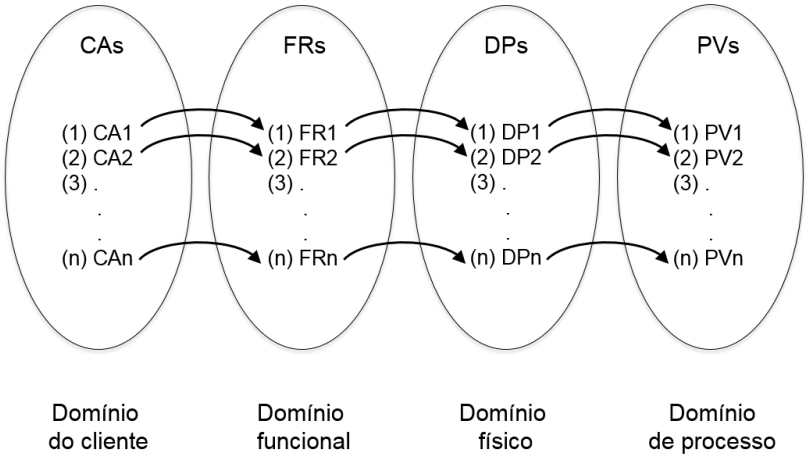


Figura 4 - Domínios do *Axiomatic Design*
Fonte: Suh (1995, 2001)

O AD inicia com a identificação da lista de requisitos funcionais (FRs), estes FRs são derivados do entendimento dos atributos dos clientes (CAs). Para cada FR encontrado é identificada uma solução ou parâmetro de projeto (DPs). Após o mapeamento do relacionamento entre FRs e DPs, o processo de decomposição utilizando *zigzagging* pode ser aplicado (ver Figura 6). O

resultado do relacionamento entre domínios é colocado no formato de matrizes de projeto (DM - *Design Matrix*). As DMs representadas na Figura 5 apresentam as relações entre cada domínio: FRs com DPs e DPs com PVs.

$$\{FRs\} = (A11 \quad A12 \quad A21 \quad A22)\ \{DPs\}$$

$$\{DPs\} = (B11 \quad B12 \quad B21 \quad B22)\ \{PVs\}$$

Figura 5 - Matrizes de Projeto
Fonte: Suh (1995)

A Figura 5 é composta por dois relacionamentos, sendo o primeiro entre FRs e DPs, no qual são utilizadas representações de quatro elementos “A” de relacionamento sendo dois FRs e dois DPs. O segundo vetor é a representação entre DPs e PVs, no qual os relacionamentos estão representados por elementos “B”. O AD apresenta mais um nível de abstração além dos requisitos e parâmetros de projeto que é chamado de variável de processo (PVs) no qual são incluídos os detalhes da implementação da solução proposta nos DPs, como métodos em uma classe de *software*.

A informação contida em cada domínio é criada por meio de um processo de análise e síntese entre domínios. A Figura 6 apresenta o processo de decomposição entre os domínios funcional e físico. A tomada de decisão inicia com o nível mais alto de forma horizontal, portanto todos os requisitos funcionais devem ser descritos sem decomposição. Em um processo de análise e síntese entre o domínio do “O que“ e o domínio do “como“ são identificados os primeiros DPs de alto nível.

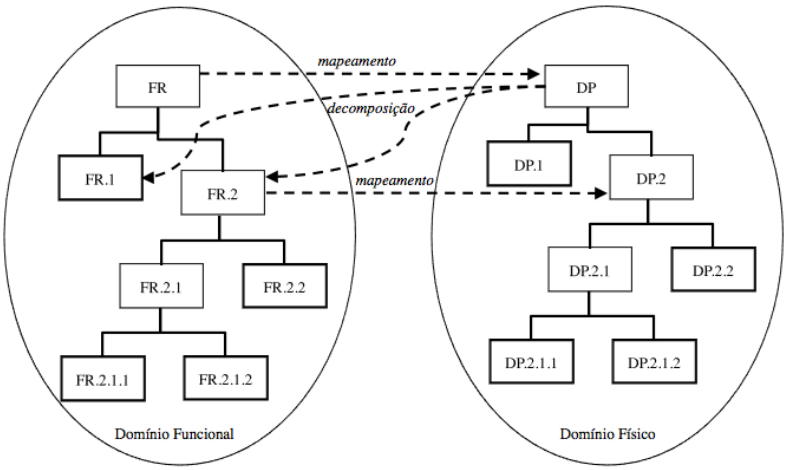


Figura 6 - *Zigzagging* Para Mapear e Decompor FRs e DPs
Fonte: Suh (2001)

Para cada FR criado é identificado um DP que pode ser lido como “a solução para satisfazer o FR”. Para decompor um FR em um vetor de características DP é necessário realizar o *zigzagging*. O *zigzagging* inicia na identificação de uma solução DP para um FR no domínio funcional (ou *zigging*). A partir da definição de um DP para o FR escolhido, volta-se (ou *zag*) para o domínio funcional realizando a decomposição e criação dos requisitos FR.1 e FR.2 que consistentemente compõem um FR superior. Na sequência para cada FR criado é necessário a identificação de DPs relativos (DP.1, DP.2) (SUH, 2001).

O resultado do relacionamento entre os domínios é representado através de matrizes de projeto (DMs). A utilização de matrizes traz para o AD uma representação matemática na qual pode ser determinado se soluções propostas (DPs) estão atendendo integralmente os requisitos (FRs) apresentados. As DMs proveem a identificação de dependências entre os domínios e autores como Suh (1990) e Pimentel (2007) representam estas dependências através de formulações matemáticas.

Através da análise das DMs resultantes, é possível avaliar o Axioma da Independência que propõe que cada requisito funcional deve ser independente. Portando, idealmente, um requisito pode ter apenas uma solução como resultado, satisfazendo o teorema 4 descrito no anexo A.2 Teoremas Gerais do AD. Caso aconteça uma modificação em um FR, então, esta modificação não deve afetar outros FRs. Caso isto aconteça, é necessária uma nova solução de projeto (Teorema 5), pois caso esta modificação impacte um FR, então, o teorema 6 e o axioma da independência estão sendo violados. O axioma da independência é aplicado sempre na presença de dois ou mais FRs, DPs ou PVs para os quais é possível fazer a análise entre as soluções propostas (SUH, 1990). Quando dependências são encontradas que violam o Axioma 1 é possível aplicar os corolários e teoremas do Anexo A. Estes teoremas trazem casos registrados de motivos para o acoplamento que violam os axiomas do AD.

A etapa seguinte do processo do AD é a verificação do conteúdo de informação em cada solução proposta. Na aplicação do axioma da informação a

solução, dentre as opções de *design* sugeridas, que envolve menor conteúdo de informação é considerada a mais indicada. A complexidade é parte da análise nesta etapa. Suh conclui que para um *design* ser considerado “bom”, este deve satisfazer os dois axiomas (SUH, 2005).

A seguir é apresentado a aplicação do axioma 1 e axioma 2 em exemplos de projeto de *software* como parte complementar na explicação sobre *Axiomatic Design*.

2.3.2 Axioma da Independência

Nesta subseção são apresentados exemplos da aplicação do axioma 1 para explicar como estes componentes do AD funcionam aplicados a projetos de *software*. Com intuito de apresentar o axioma da independência, são apresentados dois exemplos de projeto de *software* usuais, sendo eles um *software* de vendas e um *software* de controle de ponto. Nestes exemplos, primeiramente, tem-se uma matriz acoplada e no segundo exemplo uma matriz semi-acoplada. Os exemplos foram adaptados de Pimentel (2007).

Utilizando o exemplo de um *software* de vendas, Pimentel (2007) apresenta a aplicação de AD sobre o ponto de vista de um projeto orientado a objetos. Neste caso de estudo, Pimentel demonstra que a aplicação de melhores práticas de desenvolvimento de *software* não garante a escolha de um bom *design* para a solução. Na figura 9 são identificados três casos de uso (FRs) e três classes (DPs) em uma solução acoplada.

FRs \ DPs	DP.1 Classe Tela Registro	DP.2 Classe Controle Registro	DP.3 Classe Registro
FR.1 Registrar clientes	X	X	X
FR.2 Registrar produtos	X	X	X
FR.3 Registrar pedidos	X	X	X

Quadro 2 - Exemplo de Projeto de *Software* Acoplado
Fonte: Pimentel (2007)

No primeiro exemplo do Quadro 2 são escolhidos os casos de uso “Registrar clientes“, “Registrar produtos” e “Registrar pedidos”. Como solução para os casos de uso foram escolhidas três classes, seguindo o modelo MVC (modelo-visão-controlador) (PIMENTEL, 2007 apud KRASNER; POPE, 1988): classe de interface com o usuário “Tela Registro“, classe controlador “Controle Registro“ e classe de armazenamento em memória “Registro”. Na matriz resultante (DM) do mapeamento entre FRs e DPs pode ser visto um acoplamento entre todos os elementos marcados com “X” evidenciando que a utilização de técnicas e conceitos de orientação objetos não garantem um resultado satisfatório no ponto de vista de independência de componentes do projeto de *software*.

No segundo exemplo, agora utilizando um *software* de controle de ponto, Pimentel (2007) apresenta uma solução considerada semi-acoplada (Quadro 3) que pode ser considerada aceitável no ponto de vista do AD. Para determinar se a matriz é acoplada, semi-acoplada ou desacoplada Pimentel apresenta uma fórmula para cálculo da “reangularidade” (PIMENTEL, 2007, p. 64), em seu estudo sobre métricas para cálculo de independência funcional.

DPs / FRs	DP.1 Cadastrar Empregados	DP.2 Registrar Ponto	DP.3 Consultar Relatórios Gerenciais	DP.4 Consultar Relatório de Ponto do Empregado
FR.1 Cadastrar Empregado	**X**			
FR.2 Registrar Ponto	*X*	**X**		
FR.3 Consultar Relatórios Gerenciais	*X*	*X*	**X**	
FR.4 Consultar Relatório de Ponto do Empregado	*X*	*X*		**X**

Quadro 3 - Exemplo de Projeto de *Software* Semi-Acoplado
Fonte: Pimentel (2007).

Outra forma de identificar qual o nível de acoplamento da solução é visualização da DM impressa. Por exemplo no Quadro 3 pode ser visualizado a diagonal de relacionamentos entre FRs e DPs com as letras “X” em negrito. Ao mesmo tempo, é possível identificar os acoplamentos com as letras “X” em itálico. Cada “X” representa uma dependência entre a linha (FR) e coluna (DP).

De fato, uma forma de identificar se o *design* é aceitável é a utilização de fórmulas matemáticas. A partir de mais de uma solução para um conjunto de requisitos é possível calcular a reangularidade de cada solução e identificar a que tem o valor mais baixo, determinando assim, a melhor solução do ponto de vista do AD.

Nesta seção foi apresentado como é a apresentação das matrizes de projeto e como axioma 1 está relacionado com estas matrizes e incorpora um dos componentes chave do *Axiomatic Design*. Entretanto, o axioma 1 não trata da quantidade de informação que um *design* pode ter. Para isto o axioma 2 auxilia e guia o entendimento de como identificar qual *design* é melhor diante de duas ou mais possibilidades.

2.3.3 Axioma do Conteúdo da Informação

Na subseção anterior foi discutido a aplicação do Axioma da Informação e suas implicações. O processo de *design* pode produzir várias soluções aceitáveis em termos do Axioma da Independência Funcional, entretanto algumas soluções podem ser consideradas superiores a outras. O Axioma do Conteúdo da Informação fornece uma medida para um *design* e, desta forma, se torna útil para a seleção de opções de *design* aceitáveis. Adicionalmente, fornece a base teórica para análise de complexidade (SUH, 2005).

Um *design* é considerado complexo quando a probabilidade de sucesso é baixa, o que quer dizer, que a quantidade de informação para satisfazer um FR é muito alta. Isto acontece quando a tolerância de um FR é pequena, assim, requerendo uma precisão maior da solução (SUH, 2005). A noção de complexidade é relacionada a faixa de projeto (ou *design range*) de um determinado FR, em outras palavras, se a faixa de projeto é estreita será mais

difícil satisfazer o FR. A probabilidade de sucesso vem da intersecção da faixa de projeto definida pelo analista para satisfazer um FR e a capacidade do sistema de produzir aquele objetivo dentro daquela faixa especifica.

A Figura 7 apresenta os conceitos de Faixa de Projeto, Faixa de Sistema, Faixa Comum e Função densidade de probabilidade (PDF) graficamente. A probabilidade de sucesso pode ser calculada por uma Faixa de Projeto para um FR e por uma determinada Faixa de Sistema que é proposta para satisfazer um FR. O eixo vertical representa a densidade de probabilidade e o eixo horizontal pode representar um DR ou DP, dependendo do mapeamento de domínios envolvidos. A função de densidade de probabilidade do sistema está traçada sobre a faixa de sistema de um determinado FR. A faixa sobreposta entre a Faixa de Projeto e a Faixa de Sistema é chamada Faixa Comum, o que quer dizer que é a única região que satisfaz este FR.

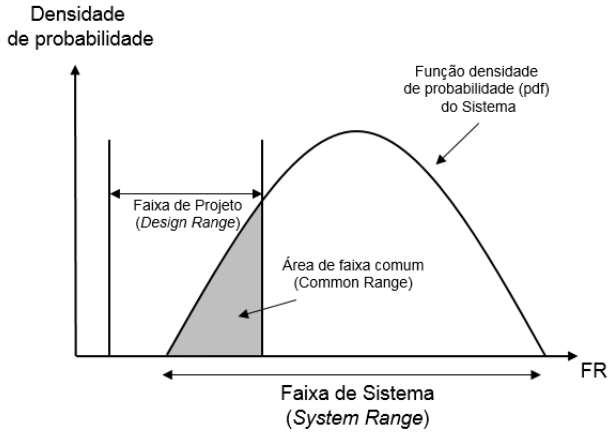


Figura 7 - Faixa de Projeto, Faixa de Sistema, Faixa Comum e PDF
Fonte: Suh (2001).

A aplicação do Axioma 2 está relacionada com a quantidade de informação contida em um *design*. Para projetos de *software* utilizam-se, por exemplo, métricas de código para calcular esta quantidade de informação. O Axioma 2 tem como objetivo diminuir a quantidade de informação necessária para o projeto (SUH, 2001). O conteúdo da informação é calculado com base na probabilidade de que o parâmetro de projeto satisfaça o requisito funcional correspondente (Pimentel 2007). Para projetos de *software*, Suh (2001) indica a métrica de linhas de código do *software* (LOC) e recomenda que quanto menor a base de linhas de código de um *software* maior é a probabilidade de realizar as funções desejadas.

Em seu trabalho, utilizando o Axioma da Informação, Pimentel (2007) recomenda uma proposta que expande a visão de Suh (2001) com a utilização de pontos por caso de uso, pontos de função e métricas CK (Métricas de Chidamber-Kemerer) para medir a quantidade de informação de um projeto de *software* orientado a objetos (PIMENTEL, 2007). No trabalho de Pimentel (2007) é utilizada uma abordagem formada com quatro etapas utilizando casos de uso, subcasos e serviços técnicos. Os subcasos são divididos em dependentes de características técnicas e independentes de características técnicas. Característica técnica é uma característica ligada à forma de realização de requisitos funcionais (PIMENTEL, 2007).

Etapa da Abordagem	**Tipo de Requisito Funcional**	**Métrica de Complexidade**
Primeira Etapa	Casos de Uso	Pontos por caso de uso
Segunda Etapa	Subcasos independentes	Pontos por caso de uso
Terceira Etapa	Subcasos dependentes	Pontos por função
Quarta Etapa	Serviços técnicos	Conjunto de métricas CK

Quadro 4 - Abordagem de Métricas de Complexidade
Fonte: Pimentel (2007).

O Quadro 4 apresenta quatro etapas da abordagem de métricas propostas por Pimentel (2007). A primeira etapa utiliza casos de uso e, como métrica, o ponto de caso de uso. A derivação de casos de uso em subcasos (uma parte de um caso de uso) independentes de características técnicas utilizam pontos por caso de uso. Subcasos dependentes utilizam pontos de função, e na última etapa, as métricas CK que compreendem: Profundidade da Herança (DIT); Número de Filhos (NOC); Acoplamento entre Objetos (CBO); Resposta para uma Classe (RFC); Falta de Coesão em Métodos (LCOM); Métodos Ponderados por Classes (WMC).

Em um caso de estudo, Pimentel (2007) apresentou a aplicação de sua abordagem através das métricas CK (exemplo ilustrado na Tabela 1) extraídas de um sistema composto por classes. No estudo da aplicação do axioma 2 foi realizada com base nas métricas propostas pelo próprio Pimentel (2007). O resultado é um índice do conteúdo (i) de informação que pode servir de

comparação entre propostas de *design* diferentes. Neste exemplo foi calculado com o valor de *i = 1,556*.

Tabela 1 - Exemplo de Valores para Métricas CK

CLASSE	WMC	EWMC	DIT	EDIT	NOC	ENOC	CBO	ECBO	RFC	ERFC	LCOM	ELCOM
CCtrlVisOpData	1	1	0	0	0	0	2	2	1	1	0	0
COpData	2	2	0	0	0	0	0	0	2	2	0	0
CBuffer	3	2	0	0	0	0	0	0	3	3	0	0
CIntDisplay	4	2	0	0	0	0	3	3	4	2	0	0
CIntKeyboard	1	1	0	0	0	0	2	2	1	1	0	0
CCtrlVisEvents	1	1	0	0	0	0	3	3	1	1	0	0
CIntFlash	2	2	0	0	0	0	0	0	2	2	0	0
COpEvent	2	2	0	0	0	0	0	0	2	2	0	0
CCtrlChangeConf	1	1	0	0	0	0	4	4	1	1	0	0
CConfiguration	4	2	0	0	0	0	0	0	4	4	0	0
CCtrlReadSerialPort	1	1	0	0	0	0	3	3	1	1	0	0
CIntRS232	2	2	0	0	0	0	1	1	2	2	0	0
CCtrlRegEvent	1	1	0	0	0	0	3	3	1	1	0	0
CCtrlReset	1	1	0	0	0	0	2	2	1	1	0	0
CIntDigInput	1	1	0	0	0	0	0	0	1	1	0	0

Fonte: Pimentel (2007).

O axioma 2 fornece uma forma de comparar propostas de *design* de alto nível através da inferência da quantidade de informação presente em cada uma das opções de *design*. A explicação sobre os axiomas no contexto de *software* é a introdução para as próximas subseções nas quais discute-se a aplicação dos fundamentos do AD da especificação de requisitos de *software*.

2.3.4 Abordagem de Especificação de Requisitos Baseado em Projeto Axiomático

A Abordagem de Especificação de Requisitos Baseado em Projeto Axiomático (AERBPA) é o tema de trabalho de dissertação de Pereira (2011) que propõe a utilização do AD dentro da especificação de requisitos de sistemas de *software*. Os componentes do AD utilizados por Pereira (2011) foram: domínios, *zigzagging*, decomposição, axioma da independência e três teoremas. Para o componente de domínio foi proposta a modificação do domínio do cliente para dois outros domínios chamados Domínio do Problema (P) e Domínio do Cliente (N) no qual residem os problemas e as necessidades dos clientes respectivamente, como ilustrado na Figura 8. Nestes novos domínios foram propostos os componentes de decomposição e *zigzagging* do AD.

Adicionalmente, foi abordado a possibilidade de integração com projetos de *software* como a proposta de Pimentel (2001).

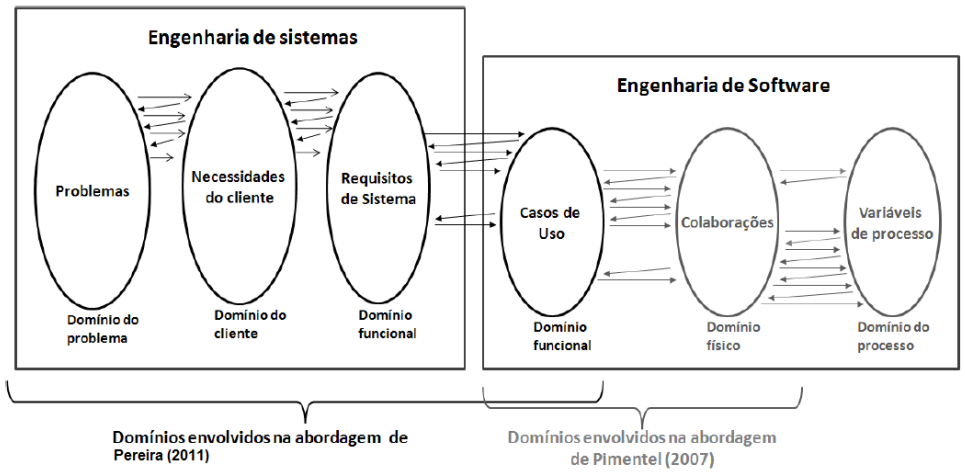


Figura 8 - Domínios da Abordagem AERBPA
Fonte: Pereira (2011).

Na proposta AERBPA foram conduzidos experimentos que demonstraram como utilizar o axioma da independência, matrizes de projeto e *zigzagging* para identificar acoplamentos, semi-acoplamento e desacoplamento. Os teoremas 1, 3 e 4 (ver a seção ”A.2 Teoremas Gerais do AD”) foram sugeridos para a análise e tomada de decisão durante o mapeamento dos domínios da Figura 8 (PEREIRA, 2011).

O processo ilustrado na Figura 9 demonstra as etapas da proposta AERBPA. A primeira etapa tem foco nos problemas e produz as matrizes de projeto (DM) para problemas. Na segunda etapa é utilizado os elementos produzidos na primeira etapa e o foco é nas necessidades, o resultado é a alteração da DM com novas necessidades, subproblemas e subnecessidades. A terceira etapa é focada nos requisitos essenciais e decompõem-se os FRs. A etapa quatro o foco é requisitos técnicos. O modelo proposto tem relação com o processo unificado proposto por Pimentel (2007).

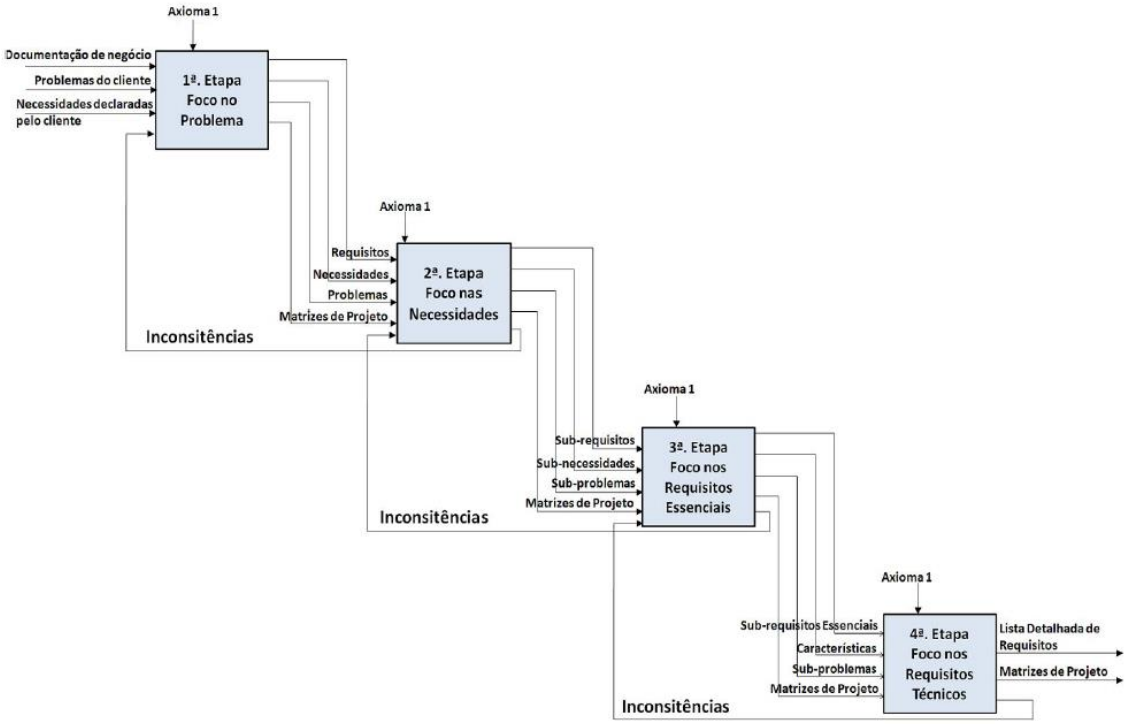


Figura 9 - Etapas da AERBPA
Fonte: Pereira (2011).

Os principais benefícios apontados por Pereira (2011) são relacionados a aspectos de processos com etapas para condução da especificação de requisitos, critérios de avaliação, demonstração da aplicação do Axioma 1 e rastreabilidade através das matrizes de projeto propostas no AD. O trabalho demonstra os potenciais de uma abordagem de requisitos de *software* utilizando AD. Entretanto, existem limitações para considerar que o AD foi aplicado de forma integral. A falta do entendimento sobre do Axioma 2 aplicado no domínio do cliente é o principal limitador daquele trabalho. Análises e casos de estudo utilizando corolários e a maioria dos teoremas são considerados importantes para entender se o AD é válido e aplicável para requisitos de *software*. O melhor entendimento das informações presentes nos domínios do problema e necessidade também é um limitador deste trabalho e será discutido nas próximas seções deste trabalho.

2.4 CONCLUSÕES

Este Capítulo apresentou a discussão sobre padrões de especificação de requisitos e o processo de Engenharia de Requisitos de *software* visto do ponto de vista tradicional, de objetivos e axiomático. Os padrões apresentados

colaboraram para o entendimento de elementos fundamentais da engenharia de requisitos como seu posicionamento dentro de outras áreas, vocabulário utilizado, características e sugestões de especificação de requisitos individuais e grupos de requisitos.

A revisão dos padrões e processos de engenharia de requisitos posiciona o trabalho em relação a normas e a compreensão da indústria. A utilização de termos semelhantes fundamenta as discussões do trabalho e entende a interpretação destes padrões. As características e sugestão de escrita de requisitos foi o ponto de início da discussão sobre o quanto estas características e escrita podem ajudar um analista de *software* a especificar com clareza e qualidade um requisito. A revisão do processo de requisitos de *software* introduziu a atual entendimento das fases que tem como saída a SRS.

Dentro dos processos e abordagens utilizadas para especificação de requisitos foram abordados rapidamente a abordagem orientada a objetivos que evidência a importância dos objetivos que originam os requisitos. Em uma compilação de vários autores, inclusive de outras áreas, a especificação de problema trouxe como é o entendimento destes autores em relação a problema e a origem de um requisito. Em uma subseção destacada foi apresentado os princípios do *Axiomatic Design* e de forma breve os trabalhos de Pimentel (2007) e Pereira (2011) que tem discussões associadas com este trabalho.

Na revisão sobre o AD foi compilado o entendimento sobre o desenvolvimento de projetos de *software* utilizando os princípios do AD e foram apresentados os elementos de processo e *design* como os axiomas, e suas principais técnicas. Em relação a especificação de requisitos foram apresentados os avanços realizados por Pereira (2011) que são utilizados na fundamentação deste trabalho.

3 MÉTODO *PROBLEM-BASED SRS*

Este capítulo apresenta a proposta de um método de especificação de requisitos de *software* baseado em problemas intitulado *Problem-Based SRS* (*Software Requirements Specification*). Este método foi desenvolvido com o objetivo de melhorar a qualidade da especificação de requisitos de *software*. Adicionalmente, são apresentadas duas técnicas complementares. A primeira avalia a qualidade da especificação produzida utilizando a teoria *Axiomatic Design*. A segunda aplica a análise de rastreabilidade de problemas e necessidades.

3.1 APRESENTAÇÃO DO MÉTODO PROPOSTO

3.1.1 Visão Geral da Proposta

O *Problem-Based SRS* é um método que tem como objetivo melhorar a qualidade das especificações dos requisitos de *software* no sentido de que os requisitos especificados fornecem respostas adequadas para as questões ou dificuldades reais do negócio do cliente. Nesta abordagem, o conhecimento (*i.e.,* a especificação) sobre os requisitos de *software* é construído a partir dos problemas do cliente. Entretanto, a abordagem trata o conceito de problemas dos clientes com uma definição diferente e estendida da utilizada comumente. A proposta é utilizar uma técnica mais precisa para entender e expressar problemas, no lugar de expressar problemas como **propósitos** (INCOSE, 2015), **necessidades** (BOURQUE e FAIRLEY, 2014) ou **objetivos** (YU, *et al.*, 2011) que não esclarecem verdadeiramente sua natureza. Os problemas são considerados como **obrigações** ou **expectativas** a respeito de questões que podem afetar os negócios do cliente. A subseção 3.1.2 discute esta definição com mais detalhes.

Mesmo que o conceito de problema proposto seja de alguma forma similar com os trabalhos revisados na seção 2, este trabalho argumenta que o conceito de problema no domínio de sistemas de *software* tem um significado não suficientemente explorado. Problemas são diferentes de “propósitos”, porque, “propósitos” representam objetivos ou intenções, e não as dificuldades originais que constituem um problema. Da mesma forma, problemas não são

“necessidades”, porque “necessidades” representam demandas ou buscas relacionadas à solução de *software* e não ao problema (*i.e.,* dificuldade) que exige uma solução. Também, problemas não são equivalentes a “objetivos”, porque “objetivos” descrevem requisitos de iniciais do *software*.

Analisando o contexto de negócio do cliente, problemas podem ser definidos e suas severidades podem ser especificadas. De acordo com a intensidade percebida dos problemas, os clientes tornam-se motivados a procurar resolver todos ou um subconjunto destes problemas (BLANK e DORF, 2012). Neste ponto, entendem-se que o cliente vislumbra possíveis tipos de soluções para os problemas. Uma das formas pode ser através de uma solução técnica como um sistema de *software* ou, também, podem tomar outras formas como contratar um novo empregado. Especificamente, este vislumbre do cliente como solução de sistema de *software* é chamado de *Software Glance* (SG) no método aqui proposto. O *software glance* não contêm muitos detalhes sobre o *software*, porém representa a solução técnica em um nível de abstração mais alto.

O *software glance* representa, em um estágio prematuro, a solução que irá fornecer o que é necessário para o solucionar os problemas do cliente. Desta forma, utilizando os problemas do cliente e levando em consideração o *software glance,* as necessidades do cliente podem ser definidas. Como será apresentado na próxima subseção, as necessidades dos clientes são definidas como saídas que o *software* deve fornecer para solucionar, ou ajudar a solucionar, os problemas do cliente. As necessidades detalham o que é a esperado como saída de um vislumbre de solução de *software* e, assim, é possível evoluir a especificação do *software glance* em uma especificação mais ampla chamada Visão de *Software (software vision)*. A seguir, o *software vision* em conjunto com as necessidades do cliente são entradas para a especificação dos requisitos. Isto significa que requisitos de *software* decorrem de necessidades e que, por sua vez, decorrem de problemas dos usuários. Há, portanto, uma dependência causal entre os problemas, necessidades e requisitos, como ilustrado na Figura 10.

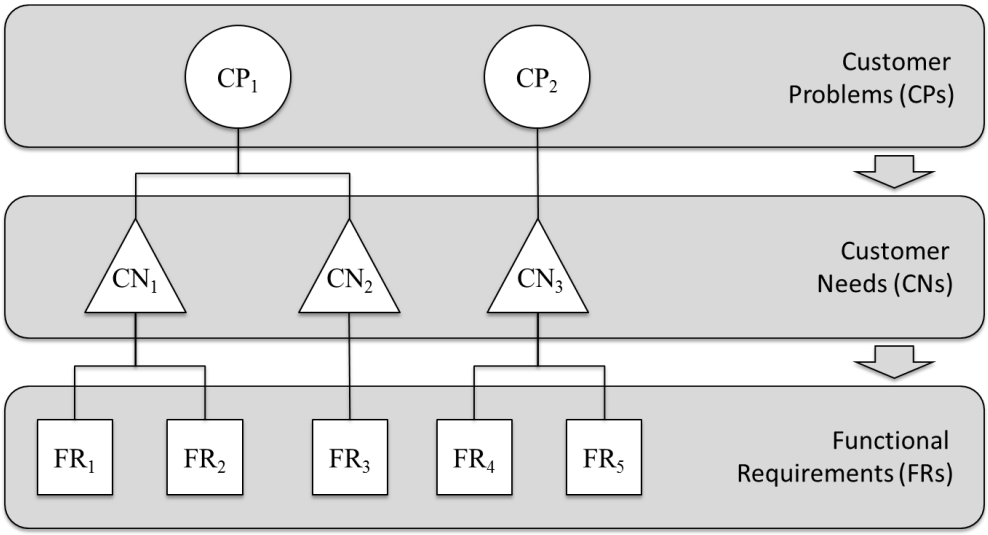


Figura 10 - Dependência Causal entre Problemas, Necessidades e Requisitos
Fonte: O Autor.

A próxima subseção apresenta os conceitos da abordagem e exemplos de cada um.

3.1.2 Conceitos Chave do *Problem-Based SRS*

Os quatro conceitos chave do método *Problem-Based SRS* são: problemas dos clientes, *software glance*, necessidades dos clientes e visão do *software*. Apesar de que estes conceitos não são inteiramente novos, neste trabalho estes conceitos tem um significado particular para o método proposto. Estes conceitos são utilizados em conjunto com conceitos como contexto de negócios e especificação de requisitos de *software*, conforme discutido a seguir.

(*i*) Problemas do Cliente - CPs

De acordo com o dicionário, um problema é uma questão que envolve dúvida, incerteza ou dificuldade (MCKEAN, 2005). No sentido matemático, um problema é uma proposição que requer uma solução. Similarmente, em engenharia, engenheiros buscam soluções para os problemas de uma empresa ou sociedade. No método proposto, considera-se que empresas tornam-se motivadas a comprar ou desenvolver uma solução quando um problema que afeta seus negócios está presente. Assim, problemas representam a razão porque empresas tentam encontrar novas soluções.

Problema é definido neste trabalho como uma questão, que pode causar ou está causando um efeito (comumente negativo) nos negócios da empresa.

Pode-se perceber problemas na forma de indicadores, medidas e métricas que mostrem quão saudável ou não está a empresa ou negócio. Dado que os problemas nem sempre são explicitamente estabelecidos de forma quantitativa, podem-se também perceber problemas como sentimentos, comumente negativos. No domínio dos negócios, estas percepções dos problemas podem tomar a forma de riscos, perda de negócios e expectativas de melhorias. Em geral, problemas são considerados coisas ruins para a empresa. Entretanto, de várias formas, problemas podem emergir das intenções de melhorar os negócios através, por exemplo, da modificação dos objetivos, seguindo novas estratégias ou adaptando tecnologias inovadoras.

Nesta pesquisa, sobre o ponto de vista do desenvolvedor de soluções de *software*, os problemas dos *stakeholders* são referenciados como como Problemas do Cliente (CPs - *Customer Problems*). Os CPs impactam no negócio de diferentes formas de acordo com a severidade do problema. Para ilustrar os CPs, considere um contexto de negócio (*i.e.,* um dado escopo de negócio) e a seguinte lista de CPs percebidos no Quadro 5. Os CPs do Quadro 5 representam exemplos de riscos e perdas percebidas no contexto de negócio. Os verbos sublinhados em cada CP evidenciam a intensidade do problema da empresa.

Definição de Contexto de Negócio:
A empresa produz e comercializa mundialmente um modelo de automóvel. Este fabricante de carros tem a intenção de adaptar o seu modelo de automóvel às novas regulamentações de emissões de poluentes. Estas regulações preocupam o negócio, porque resultam em penalidades financeiras e degradação da imagem pública da empresa, se caso não atendidas pelo modelo de automóvel. A companhia deve construir uma solução técnica para assegurar a aderência com as regulamentações de emissões sem comprometer outros aspectos técnicos do automóvel.
Definição do Problema do Cliente:
CP.1 - A modelo de automóvel atual da empresa é 80% compatível com as recentes regulamentações de emissão e desta forma a empresa deve adaptar

o automóvel para cumprir integralmente com estas regulamentações, caso contrário, poderá sofrer penalizações legais/ambientais.

CP.2 - A empresa deve entregar o modelo de automóvel adaptado às regulamentações de emissão antes das empresas concorrentes, caso contrário, irá perder a oportunidade de marketing desta atualização inovadora.

CP.3 - A empresa deve assegurar que as modificações no modelo do automóvel obedecem às regulamentações de emissões e não comprometem outros aspectos técnicos do modelo do automóvel sob pena de desagradar os clientes ou pôr em risco a qualidade ou segurança do modelo.

CP.4 - Clientes da empresa do modelo de automóvel têm a expectativa de que o modelo será o melhor em termos de redução de emissões, caso contrário, acarretará em redução da atratividade do modelo de automóvel.

CP.5 - Clientes da empresa do modelo de automóvel tem intenção de contribuir com a redução do efeito estufa e têm a esperança de que o modelo de automóvel irá encorajar outras pessoas na mesma causa, pois caso contrário, eles podem ficar desapontados.

Quadro 5 - Exemplo de Problemas do Cliente
Fonte: Autor.

(*ii*) *Software Glance* - SG

Quando os problemas dos clientes estão estabelecidos, o desenvolvedor da solução elabora um rascunho de ideia sobre um *software* que solucione ou ajude a solucionar os problemas. Esta, ideia inicial, fornece a direção da solução em termos amplos e permite identificar as principais peças do *software* vislumbrado. Por exemplo, esta ideia poderia ser: “Um sistema de informação incluindo base de dados, interface para coleta de dados de fontes na *internet* e operado por um profissional”. Esta ideia ou vislumbre de solução de *software* pode ser a solução para os problemas específicos definidos. Nesta pesquisa, referencia-se este vislumbre ou “rascunho de ideia” como *Software Glance*.

O *software glance* (*e.g.,* definição da visão do *software* e especificação das necessidades do cliente) será útil para análises e sínteses a seguir na especificação e concepção da solução de *software*. Mesmo que o *software glance* não forneça detalhes sobre a arquitetura do *software*, funcionalidades ou

comportamentos, ele representa a definição inicial da solução de *software* em alto nível de abstração.

(*iii*) Necessidades do Cliente - CNs

Dado que, por definição, os problemas dos clientes causam algum tipo de desconforto, dor, possibilidade de perda ou expectativa, o cliente (*i.e.,* quem sofre o problema) tende ou é motivado a procurar uma solução. Neste estágio inicial, a empresa vislumbra a solução (um *software glance*) e começa a definir o que é necessário nesta solução para resolver ou minimizar os problemas percebidos. Estas necessidades são referenciadas como Necessidades do Cliente (CNs - *Customer Needs*).

De acordo com Leffingwell e Widrig (2003), as necessidades do cliente (referenciadas também como necessidades dos *stakeholders*) são parte do domínio do problema e são definidas pelos usuários reais. Estes usuários reais se expressam em termos não técnicos e indicam as necessidades para resolver seus problemas usando sentenças como: “Eu preciso de um novo sistema de logística para otimizar meus custos e derrotar meus competidores”. Estas descrições simples, em linguagem natural, dizem o que o *software* deve prover para resolver os problemas.

É importante para notar que as necessidades dos clientes não descrevem funcionalidades de um *software* a ser desenvolvido. Em outras palavras, o cliente não necessita funções de um *software*. Em vez disto, cada necessidade descreve o que o *software* deve fornecer para o cliente resolver parte do problema. As necessidades são focadas nos resultados que um *software* produz ou entrega, como as informações que podem ser fornecidas.

Como exemplo, considere o problema de um gerente que precisa pagar as faturas em uma data especifica, caso contrário, a empresa terá que arcar com multas e juros. Uma solução de *software* possível para este problema é fornecer informação que avise o gerente sobre as datas de vencimentos das faturas. Consequentemente, a necessidade do cliente originada do problema especificado poderia ser: “O gerente precisa estar ciente sobre as datas de vencimento das faturas por meio das informações fornecidas por uma solução de *software*”.

O Quadro 6 apresenta alguns exemplos de necessidades de clientes relacionadas ao problema CP.4, introduzido no Quadro 5.

Definição de Necessidades do Cliente:
CN.1 - A empresa precisa de um sistema de *software* (um sítio de Internet) para permitir aos seus clientes saberem (para estarem cientes) sobre as especificações de emissão de gases do modelo de automóvel da empresa. **CN.2** - A empresa deseja um sistema de *software* para permitir ao cliente saber quão substanciais são as emissões de gases do automóvel da empresa com relação aos produtos concorrentes.
CN.3 - A empresa visa ter um sistema de *software* para permitir aos clientes saberem sobre as opiniões de outros clientes a respeito das emissões de gases do modelo de automóvel da empresa.

Quadro 6 - Exemplo de Necessidades do Cliente
Fonte: Autor.

(*iv*) Visão do *Software* - SV

O documento de visão de projeto de *software* (SV - *Software Vision*) é um documento comumente utilizado para descrever a visão alto nível de um *software* a ser desenvolvido. Este documento fornece uma visão geral do objetivo do *software* a ser desenvolvido, funcionalidades macros, o escopo e limitações, posicionamento, ambiente, *stakeholders* e outras informações gerais. Neste âmbito, o Processo Unificado de *software* da IBM (RUP) propõe descrever tais informações como um documento de visão (JACOBSON, BOOCH e RUMBAUGH, 1999) (IBM, 2015) e apresenta um modelo para a escrita do documento. Similarmente, outros autores propõem o uso do documento de visão como parte do processo de engenharia de requisitos de *software* utilizando modelos análogos ao da IBM (LEFFINGWELL e WIDRIG, 2003; WIEGERS e BEATTY, 2013).

O documento de visão contém a descrição técnica do *software* que é útil para suportar a especificação de requisitos de *software* (*SRS*). Este documento fornece os limites de escopo para o *software* e orienta a especificação de requisitos dentro deste escopo. A documento de visão contém um detalhe maior sobre a solução do que aquele apresentado no *software glance*. Ele inclui

decisões de alto-nível relativas à organização do *software* e suas conexões com o ambiente. Mesmo apresentando alguns aspectos da arquitetura, este documento não tem a intenção em descrever a verdadeira arquitetura da solução, mesmo porque, esta arquitetura será desenvolvida a seguir na fase de projeto durante o processo de desenvolvimento de *software*.

3.2 MÉTODO *PROBLEM-BASED SRS*

3.2.1 Visão do Processo de Especificação Proposto

O método *Problem-Based SRS* proposto é uma organização de atividades e artefatos (*i.e.,* documentos) resultantes. O método tem como propósito suportar o time de engenharia de requisitos, sistematicamente, analisar o contexto de negócio e especificar os requisitos de *software*. A Figura 11 apresenta um Diagrama de Objetos e Processos (OPD - *Object-Process Diagram*) que delineia o método *Problem-Based SRS*. O “APÊNDICE C - Notação *Object Process Diagram*” descreve a notação OPD utilizada.

O *Problem-Based SRS* consiste de um processo que contêm 5 atividades que também são chamadas de processos ou subprocessos, como apresentados na Figura 11. A primeira atividade é o processo de “Especificação dos Problemas do Cliente” que toma como entrada os resultados da análise do contexto de negócio e produz a lista de problemas do cliente como saída. A segunda atividade é o processo de “Projeto do *Software Glance*” visando atender aos problemas do cliente. Na terceira atividade, “Especificação das Necessidades do Cliente”, são especificadas as necessidades para resolver os problemas do cliente, levando e conta a solução vislumbrada (*software glance*). Na quarta atividade, “Projeto da Visão do *Software*”, a visão do *software* é produzida detalhando a definição de solução apresentada no *software glance* e considerando as necessidades especificadas. Finalmente, a atividade 5 é processo “Especificação dos Requisitos de *Software*” para atender as necessidades do cliente, de acordo com a visão definida para o sistema de *software*.

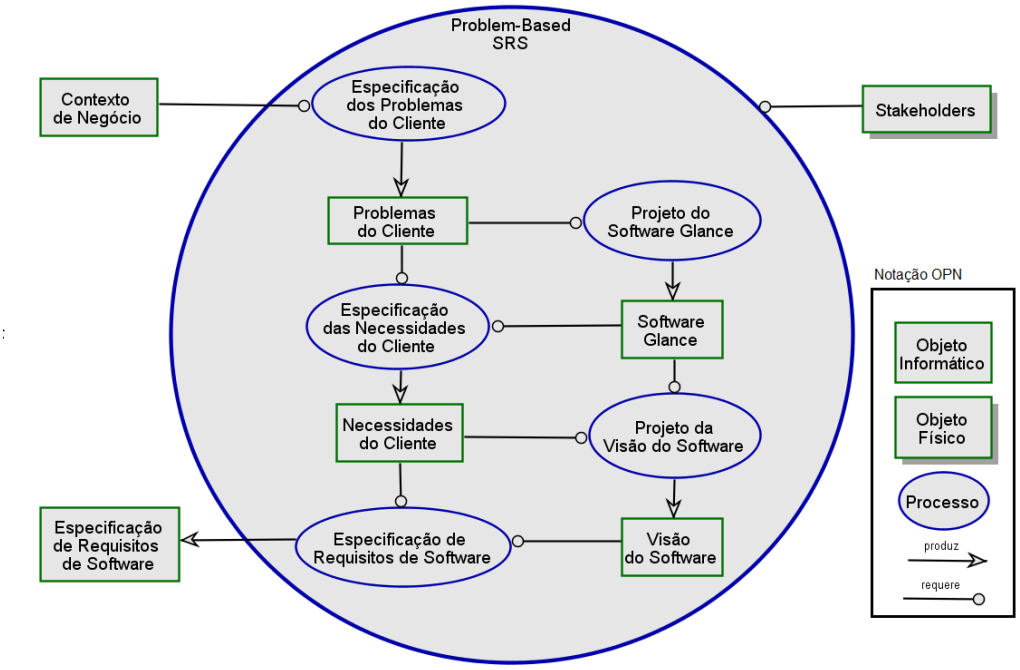


Figura 11 - Diagrama do Processo do *Problem-Based SRS* em OPD
Fonte: O Autor.

O conjunto de atividades previstas no método proposto é conduzido pelo analista de *software*. Em todas elas, deve haver a participação dos *stakeholders* que fornecerão o conhecimento sobre o contexto de negócio e percepções sobre os problemas, necessidades e eventuais restrições da solução.

O método *Problem-Based SRS* pode ser conduzido de forma sequencial ou, igualmente, de forma iterativa e incremental. Embora, a Figura 11 sugira um fluxo sequencial de processos da primeira à última atividade, o analista deve considerar que estes processos não necessariamente precisam ser sequencialmente sincronizados. Desta forma, por exemplo, a segunda etapa poderia iniciar, mesmo que a primeira não estivesse completamente concluída. O emprego de uma abordagem iterativa, permite alinhar o método proposto com as práticas mais atuais de engenharia, incluindo métodos ágeis.

A subseção a seguir detalha os processos que compõem o *Problem-Based SRS* indicando o relacionamento entre processos e suas entradas e saídas geradas.

3.2.2 Especificação dos Problemas do Cliente

O processo de Especificação dos Problemas do Cliente tem a intenção de estudar o contexto de negócio visando inferir ou determinar os CPs e escrevê-

los na forma de declarações. Para determinar os CPs, o analista de *software* pode explorar o contexto de negócios e interagir com os *stakeholders*. Ele pode fazer uso de questões como: Qual problema é evidente por si mesmo no contexto de negócio? Algum problema específico envolvendo sofrimento ou penalidade pode ser observado? Como a empresa percebe as dificuldades? Existem métricas ou índices apontando as dificuldades atuais? Existe alguma ação vislumbrada para mitigar as dificuldades encontradas? Existe algum sentimento dos *stakeholders* sobre as dificuldades no contexto de negócio? Qual é o nível de "dor" que a empresa está percebendo? Existe algum risco eminente? (OSTERWALDER, *et al.*, 2014).

Este método sugere uma nova notação para escrever declarações ou sentenças de problemas (CPs). A sentença de um CP deve ser escrita na forma de uma frase em linguagem natural em um formato padrão e técnico, incluindo um sujeito, um verbo, um objeto e uma penalidade. O sujeito declara quem sofre ou sente o problema (*e.g.,* a empresa, o *stakeholder ou* o cliente). O verbo indica a intensidade e a forma como o problema atinge o sujeito. Por exemplo, o problema pode ser uma "obrigação" indicando que ele representa um problema severo ao sujeito. O objeto do problema descreve a dificuldade que é a fonte dos problemas. Finalmente, a penalidade associada ao problema declara os custos, dores, punições ou ganhos que são consequentes do objeto do problema. Penalidades decorrentes de problemas são opcionais nas orações de problemas, porque, em algumas situações estas penalidades são evidentes.

Em relação ao verbo na definição de um problema, propõe-se três classes para escrita de CPs:

- **Classe Obrigação**

Esta classe indica que o objeto do problema é uma obrigação para o sujeito. Isto significa que o sujeito deve realizar o objeto porque uma penalidade é imposta. Esta é a classe de problema mais severa sendo que o cliente não tem outra escolha, a não ser cumprir o que é imposto. Verbos como "deve", "tem" ou "é obrigado" podem ser usados para expressar esta classe de problema. A obrigação pode vir, por exemplo, de leis, do cliente ou mercado, e de fontes legais ou padrões regulatórios. Adicionalmente, uma obrigação pode ser auto

imposta pelo próprio cliente. Neste caso, pode-se também empregar verbos como “intenciona” ou “almeja”.

- **Classe Expectativa**

Esta classe indica que há uma expectativa oriunda do cliente sobre o sujeito ou sobre o objeto. Esta expectativa não é tão severa como uma obrigação, entretanto representa uma crença forte de que o objeto será ou poderá ser realizado. Consequentemente, o sujeito pode sofrer uma penalidade por causa da expectativa considerada significativa por um cliente do sujeito.

- **Classe Esperança**

Esta classe indica que há uma esperança oriunda do cliente sobre sujeito ou sobre o objeto. Em outras palavras, uma esperança é uma possibilidade (em geral fraca) de que o sujeito irá realizar o objeto. Esta é a menos severa das classes de problemas. Entretanto, ela merece atenção porque uma esperança também pode influenciar um cliente do sujeito.

Para escrita dos problemas, propõe-se uma nova notação para as sentenças de descrição de problemas. O Quadro 7 apresenta a notação e exemplos das sentenças de problemas.

(Sujeito) (Verbo) (Objeto) (Penalidade)
Exemplo 1: Uma Empresa **(Sujeito)** deve **(Verbo)** enviar o relatório em até 10 dias **(Objeto)**, caso contrário uma multa de 10% sobre o valor total será aplicada **(Penalidade)**.
Exemplo 2: O cliente da empresa **(Sujeito)** tem a expectativa **(Verbo)** de que o tempo de espera no *callcenter* seja menor do que dois minutos **(Objeto),** caso contrário ele irá fazer uma reclamação oficial para as entidades regulatórias **(Penalidade).**
Exemplo 3:

A empresa **(Sujeito)** sofre a esperança de seus clientes **(Verbo)** de que uma ordem de compra possa ser cancelada mesmo após a sua confirmação **(Objeto),** caso contrário o cliente se sentirá desapontado **(Penalidade).**

Quadro 7 - Exemplos de Sintaxes de CPs
Fonte: Autor.

3.2.3 Projeto do *Software Glance*

O Projeto do *Software Glance* é o processo no qual o analista de *software* constrói a primeira descrição de alto nível da solução de *software*. O objetivo desta atividade é ajudar o analista a ter uma compreensão sobre as delimitações do sistema, sua arquitetura de alto nível, as principais interfaces do *software* e seus atores. Durante esta atividade, o analista de *software* pode utilizar diagramas para representação visual do *software glance*. A Figura 12 apresenta um exemplo simples do *software glance* de um sistema de vendas ilustrado na forma de um diagrama de blocos. Adicionalmente, o *software glance* pode ser documentado na forma de sentenças (como ilustrado no Quadro 8) e explicações complementares.

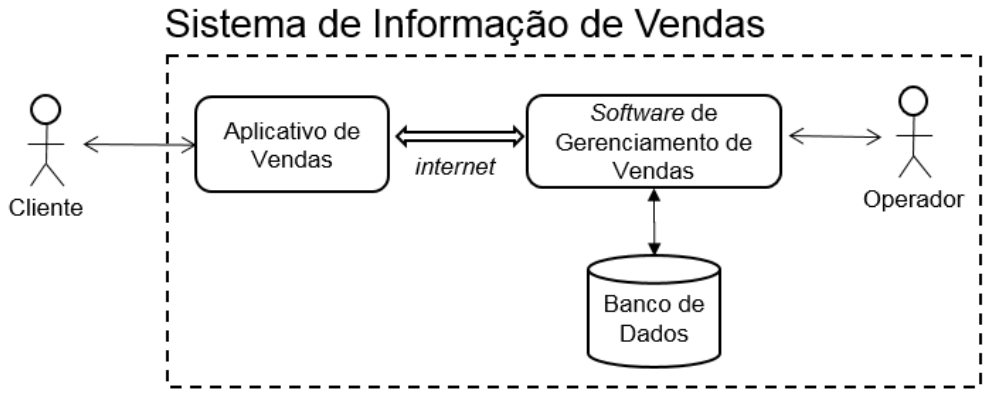


Figura 12 - Exemplo de Diagrama Ilustrando um *Software Glance*
Fonte: O Autor.

Definição do *Software Glance*: O sistema de informação de vendas irá permitir o gerenciamento das vendas de produtos originados de clientes utilizando plataformas móveis. A aplicação móvel irá prover para os clientes informações sobre os produtos. Irá, também, prover os dados das compras para o gerenciamento de vendas. Um operador irá interagir com o sistema de gerenciamento para a realização de tarefas gerais. As informações produzidas devem ser persistidas em um banco de

dados local. A comunicação entre o *software* de gerenciamento de vendas e o aplicativo móvel irá utilizar a *Internet*.

Quadro 8 - Exemplos de *Software Glance*
Fonte: Autor.

3.2.4 Especificação das Necessidades do Cliente

O processo “Especificação das Necessidades do Cliente” identifica e mapeia as CNs para satisfazer os CPs, considerando o *Software Glance*. Cada sentença de necessidade é resultado de um mapeamento direto de um problema específico, o que quer dizer que o problema representa a origem de uma necessidade particular. O *Software Glance* cria as delimitações e oferece referências do *software* para as CNs resolverem ou mitigarem o problema.

No método proposto, considera-se que uma solução de *software* pode fornecer, basicamente, quatro classes de resultados: informação, controle, construção e entretenimento. Assim, CNs são expressas utilizando estes resultados, como discutido a seguir.

Informação é um resultado comum na maioria dos sistemas de *software* em diferentes formas, como: relatórios impressos, tabelas de dados, gráficos em tela, alertas, sons, mensagens de atenção e sinais luminosos. Desta maneira, uma CN pode indicar que um cliente necessita saber, estar ciente, ter conhecimento, ser informado ou ser lembrado a respeito de alguma informação.

Sistemas de *software* também fornecem **controle**, o que significa uma supervisão e ação contínuas de acordo com uma lógica de execução. Por exemplo, em uma aeronave, há necessidade de controlar continuamente a pressão dentro da cabine utilizando sensores e um sistema de *software*.

Mais e mais, sistemas de *software* fornecem meios para **construir** artefatos digitais, como textos, modelos, desenhos, imagens e vídeos. Desta forma, a construção ou modelagem de artefatos são resultados de *software* para possíveis CNs.

Finalmente, um sistema de *software* pode fornecer **entretenimento** como resultado. Exemplos são saídas na forma de música, vídeo e jogos.

No método proposto, apresenta-se uma nova notação para escrita das sentenças de necessidades, utilizando linguagem natural e incluindo um sujeito,

um verbo, um meio, um objeto e uma condição. O **sujeito** representa quem (*e.g.,* a empresa, o gerente ou o cliente) tem a necessidade. O **verbo** indica a necessidade originada no problema utilizando verbos como querer, intencionar, desejar e precisar. O **meio** define quem fornecerá o objeto da necessidade. No contexto deste estudo, o meio sempre será o sistema de *software* alvo da especificação (delineado no *software glance*). O **objeto** representa a intenção da necessidade, ou seja, o que o meio deve fornecer em termos de informação, controle, construção ou entretenimento. A **condição** descreve restrições ao objeto da necessidade, como o período ou a precisão do objeto.

Quadro 9 apresenta a proposta de notação para as sentenças de especificação de necessidades e alguns exemplos de CNs utilizando a notação proposta.

(Sujeito) (Verbo) (Meio) (Objeto) (Condição)
Exemplo 1: O gerente **(Sujeito)** necessita **(Verbo)** um sistema de *software* **(Meio)** para saber o saldo das contas dos clientes **(Objeto)** a cada trimestre **(Condição)**.
Exemplo 2: O diretor de recursos humanos **(Sujeito)** quer **(Verbo)** um sistema de *software* **(Meio)** para estar consciente sobre as faltas dos empregados **(Objeto)** mensalmente **(Condição)**.

Quadro 9 - Notação e Exemplos de Necessidades dos Clientes
Fonte: Autor.

3.2.5 Projeto da Visão do *Software*

O Projeto da Visão do *Software* é o processo com o qual o analista e os *stakeholders* objetivam obter acordo sobre o escopo de alto nível e o posicionamento do *software*. O *software glance* fornece o ponto de início e um rascunho com as diretivas do *software*. Este processo tem, como saída, o documento de visão, discutido na subseção 3.1.1, e envolve atividades comuns (LEFFINGWELL e WIDRIG, 2003) (WIEGERS e BEATTY, 2013) como definição do posicionamento da solução de *software*, detalhamento dos *stakeholders*,

descrição geral do produto e suas funcionalidades de alto nível e definição do ambiente e restrições.

Neste trabalho foi escolhido um modelo para a visão de *software* baseado no modelo da IBM (IBM, 2015), composto pelos seguintes tópicos:

Posicionamento: o posicionamento descreve brevemente a oportunidade de negócio relacionado com o projeto de *software*. Este posicionamento utiliza os problemas apresentados pelo cliente e as informações do *software glance* como referências.

Stakeholders: são os interessados pelo projeto ou principais atores que interagem com o *software*. Alguns *stakeholders* estão presentes no *software glance*, mas, com a evolução do entendimento das necessidades, outros possíveis *stakeholders* podem ser revelados e, então, especificados neste tópico.

Funcionalidades de Alto-Nível: são as capacidades de alto-nível que o *software* deverá prover aos *stakeholders* visando entregar benefícios aos clientes. As funcionalidades são interligadas com os problemas e necessidades, porém são descritas em um nível maior de abstração. Não se deve confundir estas funcionalidades com os requisitos funcionais que são especificados posteriormente, pois funcionalidades estão relacionadas com a utilização do *software* e não com sua especificação funcional propriamente dita.

Ambiente: determina os recursos que o *software* deve ter para integração como hardware e objetos ou sistemas externos. As condições de execução e disponibilidade também podem ser especificadas neste tópico.

Adicionalmente, foi escolhido a inclusão de um diagrama de blocos para descrever uma visão arquitetural do sistema estendida do modelo do *software glance*.

3.2.6 Especificação de Requisitos de *Software*

A Especificação de Requisitos de *Software* é o processo que visa identificar claramente as exigências funcionais e não funcionais que o *software* deve atender. Este processo está relacionado à área de conhecimento, denominada Engenharia de Requisitos (BOURQUE e FAIRLEY, 2014) envolvida

com a elicitação, análise, especificação e validação dos requisitos de *software*. Este processo também inclui o gerenciamento dos requisitos durante o ciclo de vida do *software*.

Os requisitos de *software* devem atender as necessidades dos clientes e devem ter características precisas como apresentadas na ISO/IEC/IEEE 2948 (2011). Particularmente, eles devem ser completos (*i.e.,* todas as necessidades devem ser atendidas pelos requisitos) e corretos (*i.e.,* todos os requisitos devem atender alguma necessidade do cliente).

A ISO/IEC/IEEE 29148 (2011) fornece uma sintaxe para escrever requisitos de *software*. Este trabalho de pesquisa adota esta sintaxe, porém sugere a escrita das condições dos requisitos no final das sentenças, de forma que todas as sentenças iniciem com um sujeito, como ilustrado no Quadro 10. Assim, alcança-se uma melhor uniformização das sentenças.

(Sujeito) (Verbo) (Objeto) (Restrição) (Condição)
Exemplo 1: O *software* **(Sujeito)** deve ligar **(Verbo)** o bit de sinal x recebido **(Objeto)** em até 2 segundos **(Restrição)**, quando o sinal x for recebido **(Condição)**.
Exemplo 2: O sistema de radar **(Sujeito)** deve detectar **(Verbo)** o alvo **(Objeto)** no alcance de até 100 milhas náuticas **(Restrição)** no estado marítimo 1 **(Condição)**.
Exemplo 3: O sistema de *software* de faturamento **(Sujeito)** deve apresentar **(Verbo)** as faturas pendentes de pagamento dos clientes **(Objeto)** em ordem de data de pagamento **(Condição)**.

Quadro 10 - Notação e Exemplos de Especificações de Requisitos de *Software*
Fonte: Autor.

3.3 RASTREABILIDADE DE PROBLEMAS, NECESSIDADES E REQUISITOS

A rastreabilidade de requisitos é um fator de qualidade de uma *SRS* (RAMESH e JARKE, 2001). Neste trabalho é utilizada a rastreabilidade de problemas e necessidades como uma técnica de análise do método *Problem-Based SRS,* contida nos seus processos. Ela tem como objetivo verificar o grau de cobertura entre os problemas e necessidades. A compreensão da rastreabilidade de requisitos de *software* dentro dos domínios de problemas e necessidades visa auxiliar o gerenciamento e controle de mudanças.

Segundo Bourque e Fairley, a rastreabilidade de requisitos de *software* relaciona-se com a cobertura de realização, a conformidade e o impacto dos requisitos dentro do ciclo de vida do *software* (BOURQUE e FAIRLEY, 2014). Assim, algumas perguntas que fundamentam a técnica proposta são:

- **Cobertura**:

Como são conduzidas as análises de cobertura de realização considerando os problemas e necessidades dos clientes? (*i.e.,* os CPs são atendidos integralmente ou parcialmente pelas CNs? Todos as CNs são atendidas pelos FRs?);

- **Conformidade**:

Como é realizada a análise de conformidade entre problemas e necessidades e entre necessidades e requisitos? (*i.e.,* Todos as CNs são necessários e tem um CP associado?).

- **Impacto**:

Como as interdependências entre problema e necessidade impactam os artefatos da especificação de requisitos de *software*?

A seção a seguir discute questões relacionadas à análise de cobertura e conformidade. A análise de impacto não foi estudada nesta pesquisa, pois aplica-se principalmente ao gerenciamento de mudanças em requisitos que é uma atividade fora do escopo desta pesquisa.

3.3.1 Análise de Cobertura e Conformidade

Rastreabilidade é uma característica de um requisito e aparece como um importante aspecto para verificar se a definição das necessidades dos clientes foi feita de forma adequada (ISO, IEC e IEEE, 2011). No método proposto, o conceito clássico de “necessidades do cliente” foi expandido para os conceitos de “problemas e necessidades” (CPs e CNs) e, com isto, o desdobramento da rastreabilidade das origens de um requisito passa por duas etapas.

A primeira etapa (Análise de Cobertura de Problemas) consiste em verificar se as necessidades (CNs) atendem completamente ou parcialmente os problemas considerados (CPs). A segunda etapa (Análise de Cobertura de Necessidades) consiste em verificar se as necessidades (CNs) são atendidas completamente ou parcialmente pelos requisitos (FRs). Em outras palavras, a análise de cobertura no domínio do cliente preocupa-se em verificar se um problema identificado pelo *stakeholder* é atendido e em que nível, por meio do fornecimento de informação, controle, construção ou entretenimento pelo *software*. Adicionalmente, verifica-se se a necessidade é, por sua vez, atendida pelos requisitos funcionais.

A representação comum para a rastreabilidade é por meio de matrizes de rastreabilidade de requisitos (RTM - *Requirement Traceability Matrix*) (ISO, IEC e IEEE, 2011). Neste trabalho, constrói-se duas RTMs para análise de cobertura: uma RTM para a análise entre CP x CN e outra RTM para a análise entre CN x FR.

O Quadro 11 apresenta um exemplo de uma RTM CP x CN com três problemas que são solucionados por meio de três necessidades correspondentes. Observa-se que todos os três problemas estão relacionados com alguma necessidade (ou seja, não há linhas em branco), indicando que todos os problemas estão cobertos pelo conjunto das CNs especificadas. Da mesma forma, não há colunas em branco indicando que todas as necessidades estão justificadas, ou seja, cada CN contribui para a resolução de algum problema.

Uma análise equivalente é feita para verificar a cobertura das necessidades pelos requisitos construindo-se uma matriz de rastreabilidade,

com linhas representando as CNs e colunas representando as FRs. Linhas em branco indicam necessidades não cobertas pelo conjunto dos requisitos definidos. O analista de *software* deve resolver estas não conformidades encontrando uma solução para estes CPs ou CNs.

CPs \ CNs	CN.1	CN.2	CN.3
CP.1	X		
CP.2		X	
CP.3			X

Quadro 11 - Exemplo de RTM para Análise de Cobertura de Problemas
Fonte: Autor.

Neste trabalho, a RTM para análise de conformidade utiliza a notação de níveis de completude para as relações entre CP x CN e CN x FR. Foram adotados dois níveis de categorias (chamados de níveis de completude) de ligações de rastreabilidade: ligação integral ou parcial, tanto entre CPxCN quanto entre CNxFR, como apresentado na Figura 13. Para completude integral utiliza-se a letra “C” e para completude parcial utiliza-se a letra “P”. Entende-se como completude integral a situação na qual um problema (CP) é solucionado de forma completa por uma CN. A completude parcial é utilizada para indicar que a necessidade (CN) ajuda a resolver o problema, porém de forma limitada (ou seja, parcial). Assim, pode ser necessário envolver outras CNs para resolver o problema integralmente ou mais próximo de completo.

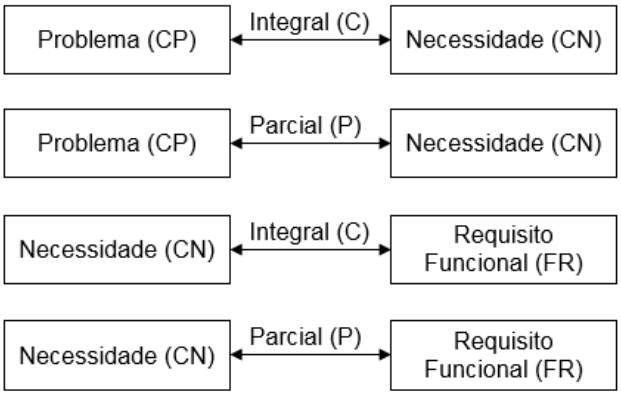


Figura 13 - Categorias de Ligações de Rastreabilidade
Fonte: O Autor.

A análise de conformidade pode ser realizada juntamente com a análise de cobertura. Esta análise, como a de cobertura, deve ser realizada em duas etapas (CPxCN e CNxFR). O objetivo da análise de conformidade é identificar se cada CP está sendo satisfeito por uma ou mais CNs e se cada CN está sendo satisfeita por um ou mais FRs. A utilização de ligações de rastreabilidade auxilia a representação da especificação de problemas, necessidades e requisitos, e a sua análise. Na representação RTM proposta, o objetivo é indicar o quanto uma informação ou controle (CN), por exemplo, pode ajudar a resolver uma obrigação ou expectativa (CP) que a empresa está sofrendo.

No exemplo ilustrado no Quadro 12, somente o problema 2 (CP.2) tem suas exigências totalmente atendidas pela necessidade 2 (CN.2). Os problemas CP.1 e CP.3 são atendidos de forma parcial pelas necessidades CN.1 e CN.3. Esta configuração deve ser discutida com os *stakeholders* para concluir se este caminho de solução e se o entendimento dos problemas e necessidades estão adequados.

CNs / CPs	CN.1	CN.2	CN.3
CP.1	P		
CP.2		C	
CP.3			P

Quadro 12 - Exemplo de RTM para Análise de Cobertura de Problemas
Fonte: Autor.

3.3.2 Exemplo de Análise de Rastreabilidade

A análise de rastreabilidade é realizada durante o desenvolvimento da SRS e busca entender aspectos de cobertura e conformidade entre problemas, necessidades e requisitos.

A etapa de análise de cobertura e conformidade dos CPs tem como objetivo avaliar se todos os problemas estão sendo atendidos e se as necessidades escolhidas ajudam a resolver o problema. O resultado desta análise auxilia o analista de *software* a identificar possíveis lacunas na resolução dos problemas dos clientes. O Quadro 12 apresenta um exemplo da primeira etapa da rastreabilidade entre CPs e CNs do caso de um fabricante de automóveis discutido na seção 3.1.1. Para este exemplo foi selecionado o

problema CP.4 que está definido no Quadro 5. As necessidades que ajudam a solucionar este problema estão definidas no Quadro 6. Neste exemplo, todas as três necessidades juntas atendem completamente (C) o problema, entretanto se analisadas em separado, cada uma ajuda parcialmente (P). Para este exemplo, foi criada uma necessidade de alto nível (CN.1) que agrupa as três necessidades (CN.1.1, CN.1.2 e CN.1.4). Em outras palavras, o cliente da empresa tem uma expectativa (CP.4) sobre o modelo do automóvel a ser produzido. Esta expectativa é atendida pelo *software glance* por meio do fornecimento de informações relevantes (CN.1) sobre a emissão de gases do modelo de veículo.

CNs / CPs	CN.1	CN.1.1	CN.1.2	CN.1.3
CP.4	C	P	P	P

Quadro 13 - Exemplo de RTM para Análise de Cobertura de Problemas
Fonte: Autor.

O resultado da análise de cobertura e conformidade do Quadro 13 indica que o CP.4 está coberto pelas necessidades e está conforme. Ou Seja, existem CNs associados ao CP.4 (assim, ele está coberto pelas necessidades) e cada CN está relacionada com o CP.4 e, portanto, elas estão justificadas. Dentro do alcance do *software* idealizado no *software glance* e especificado nas necessidades de *software* (CN.1), o problema CP.4 está completamente atendido pelas necessidades e, desta forma, não há nenhuma ação adicional por parte do analista de *software*.

A segunda etapa da análise de rastreabilidade é a análise de cobertura e conformidade das necessidades. O Quadro 15 apresenta uma RTM com a CN.1 (extraído do Quadro 6) e três FRs (Quadro 14) formuladas para atender esta CN.1. Para realizar esta análise, foi criado um FR de alto-nível (*i.g.,* FR.1) para agregar as subFRs.

FR.1 - O sistema de *software* deve apresentar uma página de internet que apresente os dados de emissão de gases do modelo do automóvel.

FR.1.1 - O sistema de *software* deve apresentar as especificações de emissão de gases do modelo de automóvel da empresa em forma de página de *internet*. FR.1.2 - O sistema de *software* deve apresentar a comparação da especificação de emissão de gases entre modelos da empresa em forma de página de *internet*.

Quadro 14 - Exemplo de FRs para a CN.1
Fonte: Autor.

FRs / CNs	FR.1	FR.1.1	FR.1.2
CN.1	C	P	P

Quadro 15 - Exemplo de RTM para Análise de Cobertura de Necessidades
Fonte: Autor.

O Quadro 15 apresenta a RTM para auxiliar na análise de cobertura e conformidade das necessidades. A análise de cobertura procura responder se os FRs conseguem atender a CN apresentada e qual é o nível de completude. Neste exemplo, a necessidade CN.1 é atendida integralmente (C) por meio da solução apresentada pelos FR.1.1 e FR.1.2. Neste exemplo, os FRs vistos de forma isolada atendem parcialmente (P) a CN.1, porém quando juntos por meio de uma agregação (*i.e.,* FR.1), o nível de completude é integral (C).

3.4 TÉCNICA DE ANÁLISE UTILIZANDO *AXIOMATIC DESIGN*

A representação da especificação de *software* por meio de domínios foi proposta por Pereira (2011) e revisada e atualizada no método proposto nesta pesquisa. A utilização dos domínios como áreas compostas por informações relacionadas hierarquicamente foi apresentada por Suh (1990) e, neste trabalho, foi estendida com relação ao entendimento sobre os domínios de *design da* abordagem Axiomatic *Design* (AD). A Figura 14 apresenta a unificação dos conceitos de especificação de *software* e o *Axiomatic Design*, apresentando os domínios de problemas (CPs), necessidades (CNs), funcional (FRs), físico (DPs) e de processo (PVs).

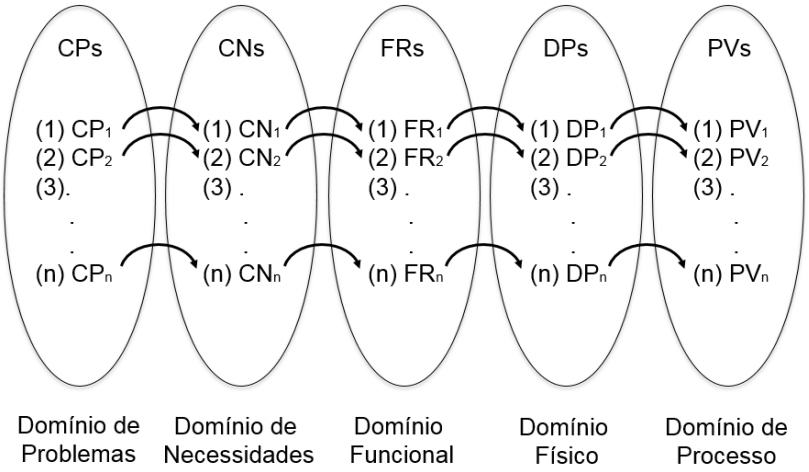


Figura 14 - Domínios do *Problem-Based SRS* e AD unificados
Fonte: O Autor.

Esta seção apresenta uma proposta de técnica de emprego de *Axiomatic Design* na especificação de requisitos e na análise da qualidade da especificação de requisitos de *software* dentro do método *Problem-Based SRS*.

3.4.1 Análise da Independência Funcional

Esta seção apresenta a interpretação do axioma da independência funcional do AD e uma técnica de análise no contexto da proposta do método *Problem-Based SRS*.

3.4.1.1 Casos de *Design*

Adaptando-se a intepretação de Suh (2005), uma especificação de *software* (SRS) pode ser considerada ideal, redundante ou acoplado dependendo da quantidade relativa de CPs, CNs e FRs. Os casos a seguir são apresentados, reformulados de Suh (2005) sob a ótica dos domínios de problemas, necessidades e requisitos.

- **Especificação Acoplada:**

(Número de CNs < Número de CPs) ou **(Número de FRs < Número de CNs)**

O Teorema 1 (APÊNDICE A - COROLÁRIOS E TEOREMAS DO AD) descreve que quando o número de elementos do domínio a direita é menor que o número de elementos a esquerda este *design* sempre é acoplado. Pode-se reescrever de duas outras formas o Teorema 1:

i) Quando o número de CNs é menor que o número de CPs, a especificação é acoplada ou não satisfaz os FRs.

ii) Quando o número de FRs é menor que o número de CNs, a especificação é acoplada ou não satisfaz as CNs.

- **Especificação Redundante:**

(Número de CNs > Número de CPs) ou **(Número de FRs > Número de CNs)**

Um *design* é considerado redundante quando a quantidade de elementos do domínio da direita é maior que a quantidade de elementos do da esquerda. Suh (2001) complementa que um *design* redundante pode ou não violar o Axioma da Independência e, assim, um *design* redundante pode ser considerado válido. O *design* redundante pode apresentar a forma de matrizes retangulares.

Adaptando-se o Teorema 3 para a especificação de requisitos, tem-se:

i) Quando existem mais CNs que CPs, a especificação é considerada redundante.

ii) Quando existem mais FRs que CNs, a especificação é considerada redundante.

Entretanto, nos dois casos, a especificação pode ser transformada produzindo em uma especificação desacoplada, semi-acoplada ou acoplada.

- **Especificação Ideal:**

(Número de CNs = Número de CPs) ou **(Número de FRs = Número de CNs)**

Uma *especificação* é considerada ideal quando o número de elementos nos domínios da esquerda e direita é o mesmo e os elementos da esquerda sempre são mantidos funcionalmente independentes entre eles. Esta matriz pode ser chamada de Quadrada ou Diagonal. O Teorema 4 pode ser adaptado para problemas e necessidades da seguinte forma:

i) O número de CNs é igual ao número de CPs, e os CPs são mantidos independentes funcionalmente entre eles.

ii) O número de FRs é igual ao número de CNs, e as CNs são mantidos independentes funcionalmente entre eles.

Dentre os três tipos de especificação mencionadas, o analista de *software* pode se deparar com uma especificação redundante, que é a mais comum e que tem algumas formas de interpretação distintas no *Problem-Based SRS*. Na teoria do AD, o *design* ideal é o melhor caso possível para uma solução e matrizes triangulares com *design* redundante são consideradas aceitáveis. Na interpretação do *Problem-Based SRS*, as matrizes triangulares são também consideradas ideais, assim como as matrizes quadradas e diagonais. O analista de *software* pode optar por reduzir as matrizes como especificado no AD para encontrar uma matriz quadrada, entretanto isto não é necessário caso o analista compreenda a relação de dependência funcional entre os domínios (*i.e.,* CPs, CNs, FRs). A utilização das matrizes de projeto e a análise de cobertura e conformidade permitem eliminar os desentendimentos que poderiam haver a respeito dos mapeamentos entre os domínios.

3.4.1.2 *Design* Redundante e Elementos Concorrentes

A compreensão se um elemento (CP, CN ou FR) é concorrente ou dependente de outro dentro do mesmo espaço de tempo é uma das questões que o analista precisa responder para melhor interpretar as dependências entre CPs, CNs e FRs. Quando dois elementos são concorrentes e um deles interfere funcionalmente no outro, esta relação é inadequada para uma especificação, pois não garante a independência funcional (*i.e.,* Axioma 1) destes elementos, criando uma dependência funcional.

	CN.1	CN.2	CN.3	CN.4	CN.5	CN.6	CN.7	CN.8
CP.1	X	X	X					
CP.2				X				
CP.3					X			
CP.4						X	X	
CP.5						X	X	X

Figura 15 - Exemplo de Matriz de Projeto de CPs x CNs.
Fonte: O Autor.

A Figura 15 apresenta um exemplo hipotético para análise, no qual exemplos é possível observar casos de dependência entre CPs e CNs que podem significar uma especificação contendo dependência funcional e necessitando uma revisão. A análise de dependências funcionais pode levar a

um resultado de matriz aceitável ou até ideal, caso não possua dependências funcionais entre os elementos.

- **Mais de um CN para um CP:**

Na linha do CP.1, ilustrado na matriz de projeto da Figura 15, é possível identificar três necessidades (CN.1, CN.2 e CN.3) resolvendo ou ajudando a resolver o CP.1. Em outras palavras, de alguma forma, as três CNs têm uma relação com o mesmo CP. A questão é se existe ou não redundância ou acoplamento nesta situação. De acordo com Teorema 3 de AD, a solução seria fazer a fusão de necessidades para eliminar a redundância ou decompor CP.1, destacando partes distintas do problema. O analista pode identificar facetas diferentes do mesmo problema e decompor o problema baseando-se, também, no conjunto de CNs proposto (conforme Corolário 1). Alternativamente, neste exemplo, as três CNs poderiam tornar-se subCNs de uma CN de alto nível, tornando assim a matriz quadrada (Teorema 4). Estas análises também são aplicadas para a relação entre CN e FR.

- **Um CN para mais de um CP:**

As CN.6 e CN.7, ilustradas na matriz de projeto da Figura 15, têm relacionamentos com mais de um problema, no caso CP.4 e CP.5. Em outras palavras, os problemas descritos têm algum tipo de dependência entre eles identificada por meio das necessidades de *software* expressas. Neste caso, é necessário que o analista faça uma análise para constatar se realmente existe dependência funcional entre estes CPs e CNs. Caso uma CN participe na solução de mais de um CP, que não sejam concorrentes (*i.e.,* que não sejam simultaneamente resolvidos pela CN), esta relação é considerada aceitável, pois não implica uma dependência funcional e, assim, não viola o Axioma da Independência. Por outro lado, caso exista dependência funcional, o analista tem a opção de desenvolver uma nova especificação (Teorema 5) ou modificar (Teorema 7) as CNs anteriormente definidos, resolvendo a dependência na resolução dos CPs. Outra possibilidade é a do analista rever os CPs (Corolário 2).

Estas análises também são aplicadas para a relação entre CN e FR. Para os dois níveis de análise do Axioma 1 de AD, podem ser aplicadas as análises de reangularidade e semangularida (PEREIRA, 2011).

3.4.2 Decomposição, *zigzagging* e hierarquia

Assim como o domínio funcional, os domínios de problemas e necessidades também podem ser decompostos hierarquicamente. Adicionalmente, um nível de hierarquia não deve ser decomposto para o próximo nível abaixo sem primeiro passar pelo domínio à direita e desenvolver uma solução que satisfaça o nível presente. Esta é a razão pela qual é necessário aplicar o processo de *zigzagging* em que se mapeia (*zig*) uma solução para o domínio da esquerda no domínio a direita e, então, se decompõe (*zag*) um novo nível na hierarquia no domínio a esquerda (SUH, 2001). Entretanto, para o método *Problem-Based SRS,* o processo de *zigzagging* não necessariamente precisa criar sempre um nível de mapeamento no domínio da esquerda no mesmo nível do da direita. Esta diferença entre os domínios, particularmente entre CPs e CNs, é devido ao nível de abstração presente neles. O domínio de problemas é um domínio de informações abstratas e de negócio, portanto pode ter limitações na decomposição de níveis mais baixos de decomposição.

A Figura 16 apresenta o conceito de *zigzagging* entre problemas, necessidades e requisitos funcionais. Neste exemplo é ilustrado que domínios à esquerda contém menos níveis de detalhe comparado a domínios a direita, pois o mapeamento de *zag* não é obrigatório. Em outras palavras, um CP pode conter várias CNs sem necessariamente conter um subCP no mesmo nível associado. Da mesma forma, uma CN pode ter vários FRs mapeados (*zig*) à direita. O processo de decomposição (*zag)* pode ser realizado caso identifique-se que é possível e desejável descrever este nível adicional de decomposição.

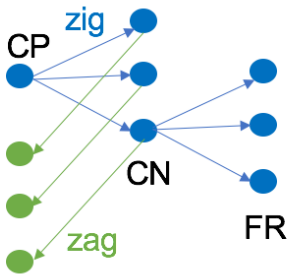


Figura 16 - *Zigzagging* para Decompor Problemas e Necessidades
Fonte: O Autor.

3.4.3 Análise do Conteúdo de Informação

De acordo com a teoria de Projeto Axiomático, o Axioma 2 determina a minimização do conteúdo de informação (CI) de um projeto, baseado na relação entre o *system range* e o *design range* (conforme a Equação 1 apresentada a seguir nesta seção). O *system range* pode ser definido pela função densidade de probabilidade de um sistema em atender um determinado FR especificado para este sistema. O *design range* estabelece a tolerância aceitável sobre um determinado FR ou a precisão desejada de um fenômeno natural a ser determinado. A função densidade de probabilidade é utilizada para representar a distribuição de probabilidade de uma variável aleatória contínua (*i.e.,* número infinito de ocorrências).

Nesta pesquisa, estudou-se a aplicação do Axioma 2 na especificação de problemas, necessidades e requisitos de *software*. A interpretação de *system range* e *design range* neste contexto é apresentada a seguir.

- **Faixa de Problema (*Problem Range*):**

Uma Faixa de Problema descreve a variação e limites de aceitação de uma solução para um determinado problema. Assim, se um problema CPn é determinado por um parâmetro Pr, a Faixa de Problema de CPn é definida pelo conjunto de valores de Pr que representam uma solução aceitável para CPn. Este conjunto de valores de Pr pode ser definido por uma função densidade de probabilidade (SUH, 2005). A Faixa de Problema pode ser entendida, também, como a faixa de tolerância de soluções para um problema.

- **Faixa de Necessidade (*Need Range*):**

Uma Faixa de Necessidade descreve a variação e limites de alcance de uma necessidade provida por um sistema para resolver um determinado problema. Assim, se uma necessidade CNn é determinada por um parâmetro Pr, a Faixa de Necessidade de CNn é definida pelo conjunto de valores de Pr que representam os resultados que podem ser produzidos pelo sistema relativos à CNn. Este conjunto de valores de Pr pode, também, ser definido por uma função densidade de probabilidade (SUH, 2005).

A Figura 17 ilustra uma relação entre a Faixa de Problema e a Faixa de Necessidade para um caso hipotético. Na situação ilustrada na figura, a Faixa de Problema equivale ao *design range*, enquanto a Faixa de Necessidade equivale ao *system range*. Como a Faixa de Necessidade cobre parte da Faixa de Problema (denominado “Área de faixa comum”), isto significa que a solução em termos de necessidades resolve parcialmente o problema. Na situação ideal, a Faixa de Necessidade coincide completamente a Faixa de Problema, presentado pela Equação 1 (NAKASAWA, 1980 apud SUH, 1990, p.309) a seguir:

$$\mathbf{I = \ln (\text{Faixa de Necessidade} / \text{Faixa de Problema}) = 0} \qquad (1)$$

Como apresentado na Equação 1, o conteúdo de informação seria igual a zero significando que a necessidade proposta resolve completamente o problema considerado.

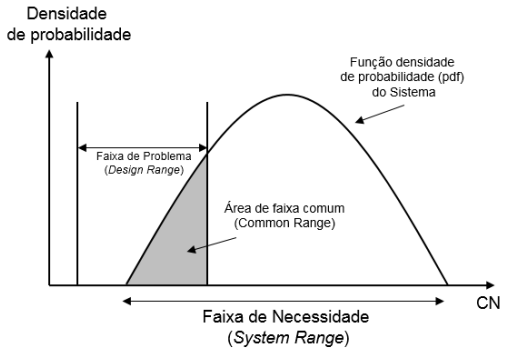


Figura 17 - Relacionamento entre Faixa de Problema e de Necessidade
Fonte: O Autor.

- **Faixa de Requisito (*Requirement Range*):**

Uma Faixa de Requisitos descreve a variação e limites de um requisito funcional a ser fornecido por um *software* para fornecer o resultado definido por uma certa necessidade. O conjunto de valores pode, também, ser definido por uma função densidade de probabilidade (SUH, 2005).

Na análise do relacionamento entre Necessidades e Requisitos, considera-se a Faixa de Necessidade como o *design range*, enquanto que a Faixa de Requisitos é considerado *o system range*, como apresentado na Figura 18. O cálculo do conteúdo de informação se faz, desta forma, com relação à Faixa de Necessidade sobre a Faixa de Requisito.

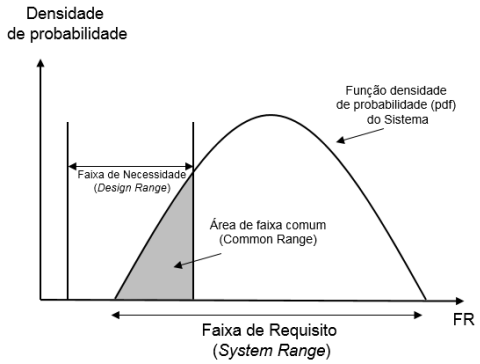


Figura 18 - Relacionamento entre Faixa de Necessidade e de Requisito
Fonte: O Autor

3.4.4 Cálculo do Conteúdo de Informação

Para ilustrar como o Axioma 2 é considerado, conforme discutido na subseção anterior, apresenta-se a seguir uma explicação da técnica de cálculo proposta nesta pesquisa.

Nakazawa (1987 apud SUH, 1990, p.307) em um estudo sobre o conteúdo da informação criou o conceito de Função de Satisfação que tem como objetivo lidar com incertezas na definição do *design range* e do *system range*. Estas incertezas estão relacionadas a características desejadas apontadas por um cliente. Para um cliente é difícil formular com precisão quais seriam os parâmetros aceitáveis para um determinado *design* principalmente em uma linguagem técnica. Contudo, um cliente pode se expressar em termos de felicidade ou satisfação sobre uma certa característica desejada. Da mesma forma, este mesmo cliente consegue descrever o que não o deixaria feliz ou o

deixaria insatisfeito. Para definir a função de satisfação, o analista deve procurar parâmetros que ajudem a delimitar o conteúdo da informação. O Quadro 16 apresenta um exemplo extraído da indústria e sua Faixa de Problema relacionado.

Problema do Cliente	Faixa de Problema
A empresa deve resolver qualquer disputa de valores de chamadas de *roaming* em até vinte dias da data na qual recebeu a notificação, caso contrário a disputa será enviada para arbitragem.	Até 20 dias

Quadro 16 - Problema e Faixa de Problema
Fonte: O Autor adaptado de GSMA (2012).

No exemplo apresentado, a empresa tem uma restrição evidente de vinte dias para resposta a um processo de disputa de valores monetários relacionados a chamadas de *roaming*. O cliente pode definir que a Faixa de Problema, ou *design range,* estaria relacionado a esta restrição de tempo. Assim, qualquer solução que entregue o resultado da disputa em até 20 dias seria aceitável e acima de 20 dias seria inaceitável devido a outros processos necessários depois da identificação da disputa. No Quadro 17 são apresentadas duas soluções alternativas (*i.e.,* CNs) contendo faixas de necessidade diferentes.

Necessidades do Cliente	Faixa de Necessidade
CN.1a - A empresa necessita de um *software* para ter conhecimento dos detalhes de disputa de tráfego de *roaming* entre 5 e 20 dias.	De 5 a 20 dias
CN.1b - A empresa necessita de um *software* para ter conhecimento dos detalhes de disputa de tráfego de *roaming* entre 10 e 15 dias.	De 10 a 15 dias

Quadro 17 - Necessidades dos Clientes e Faixa de Necessidade
Fonte: O Autor.

A Figura 19 ilustra de forma gráfica estas duas alternativas de Faixas de Necessidade indicando a cobertura da Faixa de Problema proporcionada por cada uma. Nesta representação é possível notar que ambas as soluções estão dentro da Faixa de Problema tornando assim ambas aceitáveis seguindo a formulação de (SUH, 1990).

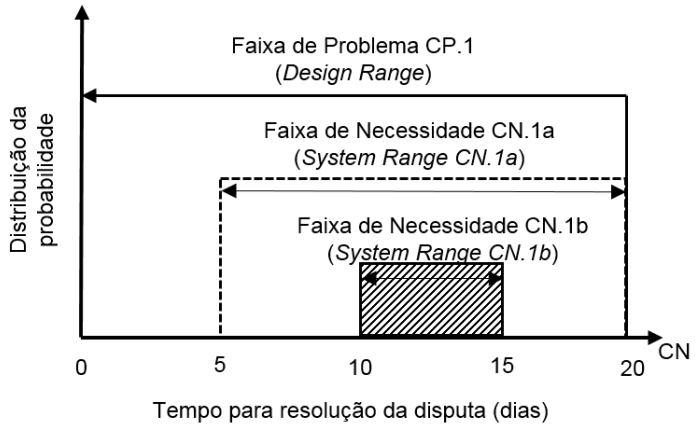


Figura 19 - Probabilidades Uniformes Como Função do Tempo para Resolução da Disputa
Fonte: O Autor

Entretanto, a Faixa de Problema é constituída de um nível de satisfação que podem variar de acordo com a quantidade de dias que se passaram desde o início do ciclo de disputa. O cliente tende a ficar mais satisfeito quanto mais rápido o conhecimento dos detalhes da disputa. A Figura 20 apresenta esta distribuição de satisfação para o CP.1, CN.1a e CN.1b. A Faixa de Problema contêm o percentual de satisfação ao decorrer dos dias que varia entre 75% e 100%, considerando que qualquer valor acima de 20 dias é inaceitável. Cada Faixa de Necessidade tem seus valores respectivos associados à Faixa de Problema.

Faixa de Problema (CP.1)

N. Dias	Satisfação
1	100%
2	100%
3	95%
4	95%
5	95%
6	90%
7	90%
8	90%
9	90%
10	85%
11	85%
12	85%
13	85%
14	80%
15	80%
16	80%
17	80%
18	80%
19	80%
20	75%

Faixa de Necessidade (CN.1a)

N. Dias	Satisfação
5	95%
6	90%
7	90%
8	90%
9	90%
10	85%
11	85%
12	85%
13	85%
14	80%
15	80%
16	80%
17	80%
18	80%
19	80%
20	75%

Faixa de Necessidade (CN.1b)

N. Dias	Satisfação
10	85%
11	85%
12	85%
13	85%
14	80%
15	80%

Figura 20 - Distribuição dos Níveis de Satisfação para CP.1, CN.1a e CN.1b
Fonte: O Autor.

De forma gráfica, a Figura 21 apresenta a distribuição de satisfação ao longo dos dias para a Faixa de Problema CP.1, demonstrando o declínio da satisfação do cliente ao longo do período.

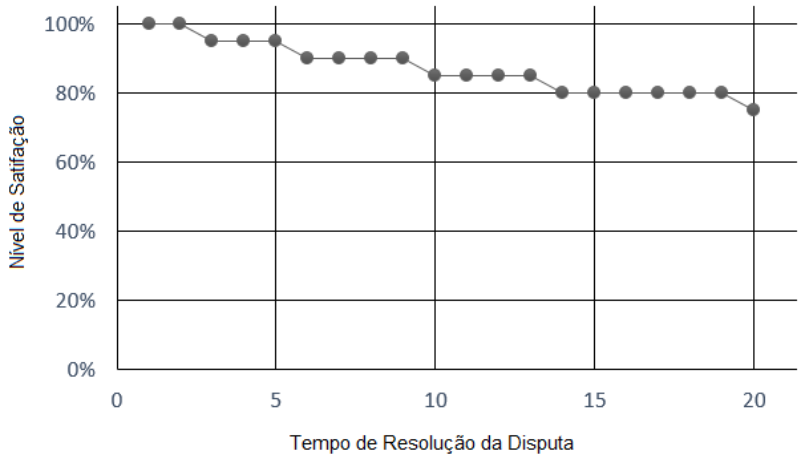


Figura 21 - Nível de Satisfação Distribuída Por Dia
Fonte: O Autor

A Figura 22 apresenta a sobreposição das faixas de CP.1, CN.1a e CN.1b utilizando os níveis de satisfação. Como resultado é possível observar a

aderência ou abrangência das duas necessidades em alcançar a expectativa de satisfação do cliente.

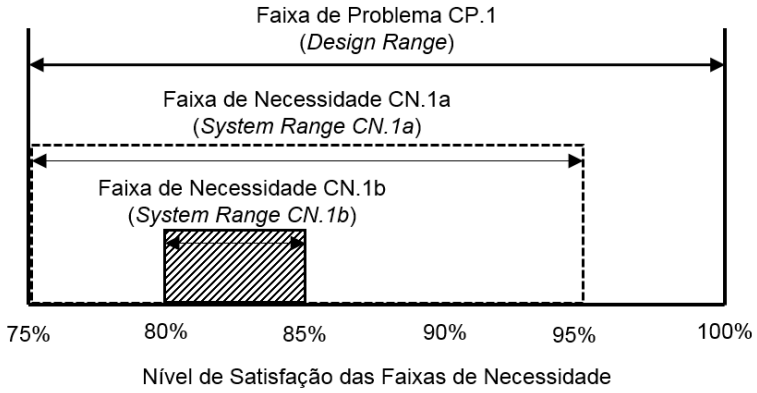


Figura 22 - Faixas de Necessidades Sobre o Nível de Satisfação do CP.1
Fonte: O Autor.

O CP associado ao tempo de resposta de resolução da disputa é de no máximo 20 dias e sua distribuição de satisfação em cada um dos dias é apresentado na Figura 20. A Tabela 2 apresenta a média ponderada de satisfação para cada uma das faixas para efeito de cálculo.

Tabela 2 - Média Ponderada Para Cada Faixa

Faixa	Média Ponderada
CP.1	87%
CN.1a	84%
CN.1b	83%

Fonte: O Autor.

Utilizando a Equação 1 de Suh (1990 apud Nakasawa, 1987), o conteúdo da informação para a opções CN.1a e CN.1b são os seguintes, expressos na Equação 2 e Equação 3:

$$\textbf{CN.1a: ln(84 / 87) = - 0,0306} \quad (2)$$

$$\textbf{CN.1b: ln(83 / 87) = - 0,0431} \quad (3)$$

A Tabela 4 apresenta o resultado do conteúdo da informação para as CNs e aponta que a CN.1a (tomando como valor absoluto) é o que apresenta o menor conteúdo de informação atendendo assim a demanda do usuário.

Tabela 3 - Conteúdo da Informação para CNs

Alternativas de CNs	Conteúdo da Informação
CN.1a	-0,0306
CN.1b	-0,0431

Fonte: O Autor.

Nesta seção identificou-se os elementos que compõem o Axioma da Informação e a proposta de cálculo utilizando o nível de satisfação para o cliente. A técnica apresentada propõe uma alternativa de interpretação e cálculo do Axioma da Informação, originalmente proposto por Suh (1990), no âmbito de Problemas e Necessidades. Esta técnica é uma forma estruturada para auxiliar o analista a quantificar a Axioma 2 em domínios não previamente discutidos.

3.5 DISCUSSÕES SOBRE O MÉTODO PROPOSTO

O método *Problem-Based SRS* proposto e apresentado neste capítulo traz quatro maiores contribuições ao processo de especificação de requisitos de *software*, conforme discutido a seguir.

a) Diferenciação entre problema e necessidade

O método *Problem-Based SRS* propõe conceitos mais precisos para “problemas do cliente” e “necessidades do cliente”. Como discutido nos trabalhos relacionados, o entendimento dos termos “problema e necessidade” geralmente é confuso e ambíguo tanto na literatura científica e técnica, quanto no cotidiano das empresas de *software*. Desta forma, os conceitos propostos contribuem para o enriquecimento da compreensão e, consequentemente, da precisão das especificações de requisitos de *software*, conforme apontado no Problema 1, discutido na introdução desta dissertação. Isto inclui, especialmente, uma possibilidade muito maior de o analista mapear adequadamente as demandas dos clientes aos requisitos que um *software* deverá atender. As chances de sucesso dos empreendimentos de desenvolvimento de *software* podem, assim, ser aumentadas consideravelmente uma vez que os problemas e necessidades dos clientes podem ser expressos em classes bem definidas. Adicionalmente, estas classes seguem notações de escrita que auxiliam o analista a criar

uniformidade na especificação de problemas e necessidades em forma de linguagem natural.

b) Abrangência do método proposto

O *Problem-Based SRS* é um método de especificação de requisitos ab*rang*ente por meio da organização sistemática das atividades, especificação dos artefatos e o fluxo lógico. O método propõe a consideração dos domínios de problema, necessidades e requisitos dentro de seu fluxo de atividades resultando na criação de uma especificação de requisitos de *software* de qualidade respondendo ao Problema 3. Adicionalmente, o método não prescreve um fluxo de trabalho específico porque as atividades podem ser conduzidas sequencialmente ou seguindo fluxos iterativos ou incrementais (inclusive como aquele dos Métodos Ágeis). Desta forma, o método proposto pode ser usado em praticamente qualquer contexto de desenvolvimento de *software*.

c) Formas de análise da qualidade da especificação

Utilizando rastreabilidade e fundamentos do AD, o método *Problem-Based SRS* responde à pergunta do Problema 2 propondo uma forma de análise que auxilia o analista de *software* durante a especificação garantindo a qualidade, integridade e precisão das especificações dos problemas, das necessidades e dos requisitos. A utilização de AD, mesmo de forma ainda limitada, auxilia o analista no entendimento da independência e da complexidade (*i.e.,* conteúdo da informação) dos problemas, necessidades e requisitos de *software*. AD fornece um arcabouço teórico bem fundamentado para a análise da qualidade da especificação de requisitos.

d) Conceito de *Software Glance*

O método *Problem-Based SRS* propõe a atividade explícita de construção da primeira abstração da solução de *software* denominada *Software Glance*. Ela representa o que o analista de *software* tem na mente em termos de definição abstrata da solução de *software* após ou ao longo do entendimento dos problemas do cliente. O termo "*glance*" equivale ao termo "vislumbre" no português, fazendo referência a uma ideia indistinta, conjetura, suposição ou hipótese sobre uma solução de *software*. Assim, o *software glance* fornece uma direção antecipada para os analistas de *software* na especificação das

necessidades do cliente e, posteriormente, para a construção da “visão de *software”*. Este conceito de *software glance* se alinha com a hipótese de que o desenvolvimento de uma solução envolve uma interação contínua entre função e forma, que também coincide com a argumentação de Nan P. Suh (SUH, 1990) de que atividade de *design* envolve uma contínua inter-relação (*interplay*) entre o que se deseja alcançar e como se pretende alcançá-lo. Entretanto, até o limite das pesquisas bibliográficas desta dissertação, nenhum conceito similar havia sido proposto.

4 ESTUDO DA APLICAÇÃO DO MÉTODO *PROBLEM-BASED SRS*

Este capítulo apresenta um estudo da aplicação do método *Problem-Based SRS* em um sistema de Gestão de Relacionamento com Clientes ou *Customer Relationship Management* (CRM). As próximas subseções são organizadas de acordo com as atividades do processo definido na seção 3.2 e descrevem brevemente como o analista de *software* pode conduzi-las.

4.1 CONTEXTO DE NEGÓCIO

O contexto de negócio é um conjunto de informações que caracteriza uma porção de um domínio de negócios e define o escopo para condução de ações como a análise, modificação e melhoria dos processos de negócio. O contexto de negócio pode ser descrito utilizando diferentes abordagens de modelagem como diagramas de contexto, árvores de funcionalidades (WIEGERS e BEATTY, 2013), sentenças em linguagem natural, BPMN, UML e modelos baseados em ontologias (NOVAKOVIC e HUEMER, 2014).

Para este estudo da aplicação do método proposto em um sistema CRM, o contexto de negócio foi definido na forma de uma sentença em linguagem natural apresentada no Quadro 18.

A empresa tem grandes dificuldades para manter um relacionamento efetivo com seus clientes e ela está convencida de que um sistema de informação como um CRM pode contribuir para redução destas dificuldades.

Quadro 18 - Contexto de Negócio para o CRM
Fonte: O Autor.

Esta breve descrição do contexto de negócio (Quadro 18) indica que a empresa está sofrendo “grandes dificuldades”, indicando que problemas podem existir no escopo de negócios que envolvem relacionamentos com seus clientes e estes problemas podem ser significativos para o negócio. Este desconforto frente às dificuldades motiva a empresa a procurar uma solução de *software* para ajudar a solucionar estas dificuldades. Finalmente, os *stakeholders* têm a percepção de que um sistema de CRM pode ser um tipo de solução adequada, talvez porque eles já conheceram outras empresas utilizando este tipo de tecnologia.

4.2 PROBLEMAS DO CLIENTE

O analista de *software* inicia o processo “Especificação dos Problemas do Cliente” conduzindo uma análise do contexto de negócio para determinar os problemas relacionados às dificuldades reportadas no contexto de negócio de CRM. Baseado no contexto de negócio, o analista pode identificar a origem dos problemas que, para o estudo considerado, é a “dificuldade de relacionamento com seus clientes”. Diferentes técnicas podem ser utilizadas para determinar os problemas dos clientes e quais são as fontes que originaram estes problemas. No método *Problem-Based SRS*, os problemas são estudados buscando-se identificar as obrigações, expectativas, esperanças e penalidades associadas às dificuldades encontradas no contexto de negócio definido.

Neste estudo da aplicação do método, o analista poderia, por exemplo, elaborar questões aos *stakeholders,* como*:*

- Como a empresa atualmente gerencia seu relacionamento com os clientes?
- Existem políticas da empresa sobre o relacionamento com os clientes?
- Quais obrigações a empresa possui no relacionamento com seus clientes?
- Existem regulamentações particulares a respeito do relacionamento com os clientes?
- A empresa conhece as expectativas e esperanças de seus clientes?
- Existem estatísticas sobre os *feedbacks* dos clientes?

Estes tipos de perguntas podem ajudar a identificar os problemas associados ao contexto do negócio e que traduzem de maneira mais precisa o que o cliente denominou “grandes dificuldades” no contexto de negócio. O Quadro 19 apresenta os CPs especificados para o CRM.

CP.1 - A empresa deve assegurar a existência de um canal de comunicação com todos os clientes, sob pena de perder clientes afetando seu *marketing*, promoções, *feedbacks* e vendas futuras.
CP.1.1 - A empresa deve garantir ser capaz de contatar todos os seus clientes.

CP.1.2 - A empresa deve assegurar que cada cliente seja contatado regularmente.
CP.2 - A empresa deve considerar as estatísticas sobre os *feedbacks* dos clientes em seu planejamento, sob pena de criar insatisfação nos clientes e consequente perda de vendas e fatia de mercado.
CP.2.1 - A empresa deve garantir que os *feedbacks* dos clientes estão sendo considerados.
CP.2.2 - A empresa deve assegurar que os cálculos estatísticos estão atualizados.
CP.3 - Os clientes têm expectativa de que a empresa responda seus *feedbacks*, caso contrário os clientes se tornarão frustrados e a empresa pode sofrer diminuição de sua reputação.
CP.4 - A empresa deve alinhar suas estratégias de venda e campanhas com o comportamento dos clientes, caso contrário a empresa corre o risco de perder oportunidades de vendas e falhar em suas estratégias.
CP.5 - A empresa deve projetar as vendas, sob pena de perder oportunidades e fazer provisões inadequadas.

Quadro 19 - Problemas para o CRM
Fonte: O Autor.

O problema CP.1 foi particionado em subproblemas: assegurar que todos os clientes possam ser contatados (CP.1.1) e serão contatados regularmente (CP.1.2). Estas duas preocupações detalham melhor o problema que o cliente está sofrendo relacionado à garantia de uma comunicação efetiva no relacionamento com seus clientes. O CP.2 particiona o problema relacionado a considerar as estatísticas dos clientes em garantir que todos os feedbacks estão sendo considerados (CP.2.1) e que os cálculos estatísticos estão atualizados (CP.2.1). Os CP.3, CP.4 e CP.5 não foram particionados em subproblemas.

4.3 *SOFTWARE GLANCE*

O *software glance* neste estudo do CRM descreve a ideia inicial ou vislumbre do *software* para enfrentar os problemas (CPs) identificados no Quadro 19. Esta é a primeira representação da solução de *software*. Neste estudo, o contexto de negócio claramente sugere um *software* de CRM como uma solução vislumbrada pelos *stakeholders*. Entretanto, *softwares* CRMs não seguem um padrão e, desta forma, necessitam ser customizados em diferentes formas. Dado que o CP.1 aponta um problema associado aos canais de comunicação com os clientes da empresa, o analista de *software* considerou que a solução poderia ter uma interface *web* com os clientes para *marketing*, promoções e receber *feedbacks* (em destaque no CP.3). CP.2, por sua vez, referência o problema de considerar as estatísticas sobre os *feedbacks* dos clientes. Isto fez o analista considerar uma interface local com o Gerente para fornecer estas estatísticas. O analista também considerou a possibilidade de analisar o comportamento do cliente (CP.4) e ser capaz de fazer previsões (*forecast*) de vendas futuras (CP.5). O *software* deve ter um banco de dados para registrar as interações com clientes e seus históricos de vendas. Adicionalmente à interface para interagir com o Gerente, o *software* deverá ter uma interface LAN (*Local Area Network*) com o *Software* de Gerenciamento de Vendas. O Quadro 20 apresenta a descrição textual do *software glance* definido para o estudo do *software* de CRM e a Figura 23 apresenta um diagrama de blocos para o *software glance*.

O *software* CRM irá interagir com os clientes através de uma interface *web* possibilitando campanhas de *marketing*, recebimento de *feedbacks* e envio de respostas para os clientes. Adicionalmente, o *software* de CRM irá fornecer interfaces locais para interagir com o Gerente. Dados sobre os clientes, *feedbacks* e o histórico de vendas serão armazenados em uma base de dados local. O *software* de CRM incluirá, também, uma interface LAN para o *software* de gerenciamento de vendas.

Quadro 20 - Exemplo de Definição do *Software Glance* para o CRM
Fonte: O Autor.

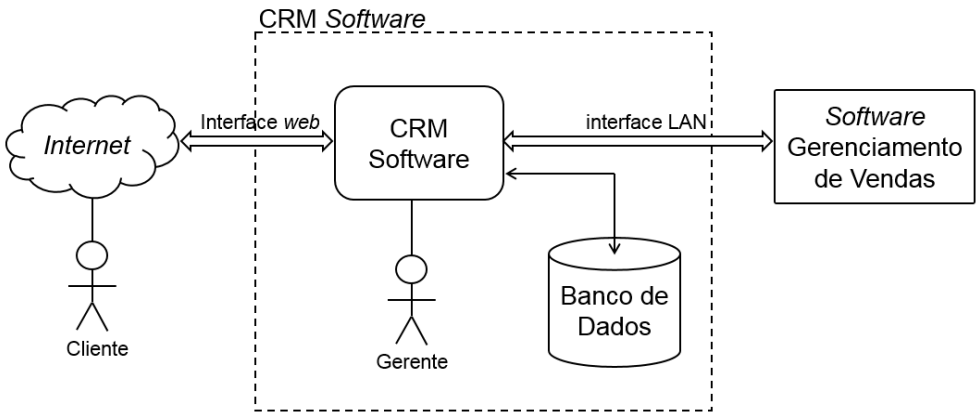


Figura 23 - Diagrama de blocos do *Software Glance* do CRM
Fonte: O Autor

4.4 NECESSIDADES DO CLIENTE

Neste ponto, o analista de *software* tem conhecimento dos CPs identificados e dispõe do *software glance* do CRM. O próximo passo é determinar o que o cliente precisa como resultados ou saídas fornecidas pela solução de *software* para resolver os CPs. No método proposto, as necessidades são expressas como informações, controles, entretenimentos ou construções que o *software* deve fornecer.

Considerando CP.1, o analista concluiu que a empresa precisa saber quem são seus clientes e necessita estar ciente sobre as informações atuais destes clientes. Adicionalmente, o analista observou que para assegurar um canal de comunicação com os clientes, a empresa precisa contatá-los regularmente e estar ciente do que está acontecendo neste relacionamento cliente-empresa. Assim, o analista especificou as necessidades CN.1, CN.2 e CN.3 como resultados necessários a serem fornecidos pelo *software* CRM para poder resolver ou ajudar a resolver o CP.1. O Quadro 21 apresenta as CNs resultantes da análise sobre os CPs.

CN.1 - A empresa necessita um *software* CRM para ter conhecimento dos clientes com canais ativos e não ativos.
CN.1.1 - A empresa precisa de um *software* CRM para identificar seus clientes e ter as informações de contatos deles.

CN.1.2 - A empresa precisa de um *software* CRM para estar ciente sobre os clientes a serem contatados para que seja mantida uma comunicação ativa com eles.

CN.2 - O Gerente do CRM necessita um *software* CRM para ter conhecimento sobre as estatísticas de *feedbacks* dos clientes.

CN.2.1 - O Gerente de Contas necessita um *software* CRM para saber sobre os *feedbacks* a serem respondidos.

CN.2.2 - O Gerente de Contas necessita um de um *software* CRM para saber as estatísticas atualizadas dos *feedbacks*.

CN.3 - O Gerente de Contas necessita um de um *software* CRM para saber sobre os *feedbacks* recebidos dos clientes.

CN.4 - O Gerente de Contas objetiva um *software* CRM para ter conhecimento sobre o comportamento dos clientes.

CN.5 - O Gerente de Contas necessita um *software* CRM para saber as projeções de vendas.

Quadro 21 - Necessidades dos Clientes para o CRM
Fonte: O Autor.

4.5 VISÃO DO *SOFTWARE*

Após especificar as necessidades dos clientes e o “vislumbre” sobre a solução de *software*, o analista de *software* pode produzir (*i.e.,* conceber ou projetar) a visão do *software* de CRM. Esta visão de *software* aumenta o detalhe fornecido no *software glance* levando em consideração as CNs do cliente. O Quadro 22 apresenta a especificação da visão do *software* que inclui os seguintes detalhes: o posicionamento do *software*, *stakeholders* envolvidos, funcionalidades de alto-nível, ambiente de *software* e um diagrama de alto nível da arquitetura do *software* (ilustrado na Figura 24).

Posicionamento: o *software* de CRM destina-se aos gerentes de conta e gerenciamento dos relacionamentos com os clientes que devem manter um relacionamento próximo com os clientes da empresa. O *software* de CRM é uma plataforma intuitiva que assegura canais de comunicação contínuos com

os clientes e auxilia no entendimento do comportamento dos clientes e na projeção das vendas.

Stakeholders: Gerente de Conta, Gerente do CRM e os Clientes da empresa.

Funcionalidades de Alto-Nível:

- O *software* de CRM deve fornecer canais de comunicação com os clientes através da *internet*.
- O *software* CRM deve fornecer previsões de vendas, analises sobre o comportamento dos clientes e estatísticas sobre os *feedbacks* dos clientes.
- O *software* de CRM deve garantir os padrões apropriados de segurança.
- O *software* de CRM deve ser compatível com as tecnologias atuais adotadas pela empresa.
- O *software* de CRM deve ser estruturado em subsistemas de *front-end* e *back-end*.

Ambiente: os Gerentes de CRM e os Gerentes de Contas estão localizados no escritório da empresa utilizando computadores padrões da empresa. A empresa disponibiliza comunicação de alta velocidade entre todos os escritórios da empresa.

Quadro 22 - Visão do *Software* para o CRM
Fonte: O Autor.

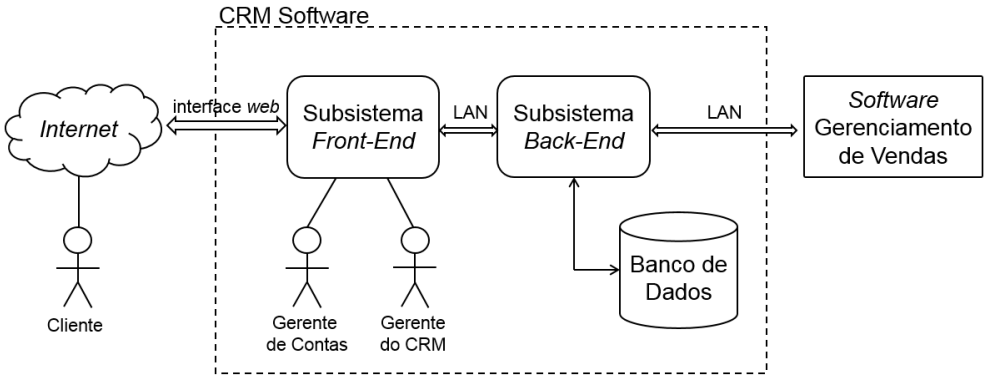


Figura 24 - Visão do *Software* CRM
Fonte: O Autor

4.6 REQUISITOS DE *SOFTWARE*

A especificação de requisitos de *software*, aplicando o método *Problem-Based SRS,* utiliza a visão do *software* e as necessidades dos clientes para definir os requisitos funcionais e não-funcionais para o *software*. No estudo do CRM, o analista precisa assegurar a cobertura completa e correta das necessidades especificadas pelo cliente no Quadro 21 e delineadas na visão do *software* no Quadro 22.

Para atender a CN.1 (*i.e.,* identificar os clientes da empresa e suas informações de contato), foram criadas as quatro seguintes FRs: cadastrar um novo cliente (FR.1), modificar dados do cliente (FR.2), visualizar os dados do cliente (FR.3), e realizar pesquisas dos clientes no banco de dados (FR.4).

Para a CN.2 (*i.e.,* estar ciente sobre os clientes a serem consultados), o analista especificou as seguintes FRs: registrar um novo contato feito com um cliente (FR.5) e visualizar os contatos realizados com os clientes (FR.6).

O Quadro 23 apresenta a lista completa dos FRs especificados para atender as CNs do Quadro 21.

FR.1 - O *software* CRM deve possibilitar ao Gerente de Contas e o Gerente de CRM gerenciar os clientes.
FR.1.1 - O *software* CRM deve possibilitar ao Gerente de Contas registrar um novo cliente no banco de dados.
FR.1.2 - O *software* CRM deve possibilitar ao Gerente de Contas modificar dados dos clientes no banco de dados.
FR.1.3 - O *software* CRM deve possibilitar ao Gerente de Contas e o Gerente de CRM visualizar dados do banco de dados dos clientes.
FR.2 - O *software* CRM deve possibilitar ao Gerente de Contas e o Gerente de CRM gerenciar *feedbacks*.
FR.2.1.1 - O *software* CRM deve permitir ao Gerente de Contas consultar os *feedbacks* relacionados a um cliente.
FR.2.1.2 - O *software* CRM deve possibilitar ao Gerente de Contas cadastrar um novo contato feito no banco de dados do cliente.

FR.2.1.3 - O *software* CRM deve possibilitar ao Gerente de Contas e o Gerente do CRM visualizar os contatos realizados e a serem realizados com clientes
FR.2.2 - O *software* CRM deve possibilitar ao Gerente de Contas e Gerente do CRM visualizar as estatísticas relacionadas aos *feedbacks* dos clientes.
FR.3 - O *software* CRM deve possibilitar o Gerente de Contas e o Gerente de CRM realizar pesquisas sobre os *feedbacks*.
FR.4 - O *software* CRM deve permitir ao Gerente de Contas e ao Gerente de CRM gerenciar perfis de comportamento.
FR.4.1- O *software* CRM deve permitir ao CRM Manager visualizar informações agrupadas por perfis de clientes.
FR.4.2 - O *software* CRM deve permitir ao Gerente do CRM cadastrar o perfil de um grupo de clientes.
FR.4.3 - O *software* CRM deve possibilitar ao Gerente de Contas e ao Gerente de CRM registrar comportamentos de um cliente.
FR.4.4 - O *software* CRM deve permitir ao Gerente de Contas e ao Gerente de CRM cadastrar o perfil de comportamento de um cliente.
FR.4.5 - O *software* CRM deve permitir ao Gerente de Contas e ao Gerente de CRM visualizar clientes com perfis de comportamento.
FR.5 - O *software* CRM deve permitir ao Gerente de Contas e ao Gerente de CRM visualizar a projeção de vendas.

Quadro 23 - Especificação de Requisitos de *Software* para o CRM
Fonte: O Autor

4.7 ANÁLISE DE RASTREABILIDADE

A análise de rastreabilidade tem como objetivos a análise da cobertura e conformidade dos problemas e das necessidades do cliente. O analista de *software* precisa verificar se todos os problemas foram tratados e se estes problemas foram solucionados. A mesma análise deve ser realizada para as necessidades verificando se existem soluções (*i.e.,* requisitos funcionais) para cada uma das necessidades apresentadas e se todas são justificadas.

A Figura 25 apresenta a matriz de rastreabilidade para a análise de cobertura e conformidade dos cinco problemas de alto nível. Os problemas CP.1, CP.2 e CP.4 foram particionados em subproblemas, de forma que cada subproblema descreve uma porção do problema de mais alto nível. Todos os problemas do estudo foram considerados completos (C), o que significa que todos estão sendo tratados de forma integral pelas necessidades. Os subproblemas, em relação ao problema de alto nível, são sempre incompletos (P) se observados individualmente. Em relação à análise de conformidade, todos os CPs são mapeados em alguma necessidade e todas as necessidades possuem relevância para (*i.e.,* ajudam a resolver) algum problema.

	CN.1	CN.1.1	CN.1.2	CN.1.3	CN.2	CN.2.1	CN.2.2	CN.3	CN.4	CN.5
CP.1	C									
CP.1.1		P								
CP.1.2			P							
CP.1.3				P						
CP.2					C					
CP.2.1						P				
CP.2.2							P			
CP.3								C		
CP.4									C	
CP.5										C

Figura 25 - RTM para Análise de Cobertura de Problemas do CRM
Fonte: O Autor.

A Figura 26 apresenta a RTM para a análise de cobertura e conformidade das necessidades de *software* do cliente, que mostra as necessidades que envolvem mais de um requisito funcional para atendê-las. Esta relação 1-para-N entre CNs e FRs é devida à transição de um domínio mais abstrato para um domínio menos abstrato ou mais próximo do domínio físico. A análise de conformidade pode iniciar com as CNs que não têm o mesmo nível de decomposição que os FRs (*i.e.,* CN.2.1 e CN.4). Nestes casos é importante verificar se cada um dos FRs associados não traz alguma forma de redundância que pode ser eliminada. No estudo de aplicação do CRM, a análise de conformidade não apresentou nenhuma correção necessária e, assim, todos as CNs estão classificados como “conformes”.

	FR.1	FR.1.1	FR.1.2	FR.1.3	FR.2	FR.2.1	FR.2.1.1	FR.2.1.2	FR.2.1.3	FR.2.2	FR.3	FR.3.1	FR.3.2	FR.4	FR.4.1	FR.4.2	FR.4.3	FR.4.4	FR.4.5	FR.5
CN.1	C																			
CN.1.1		P																		
CN.1.2			P																	
CN.1.3				P																
CN.2					C															
CN.2.1						P	P	P	P											
CN.2.2										P										
CN.3											C	P	P							
CN.4														C	P	P	P	P	P	
CN.5																				C

Figura 26 - RTM para Análise de Cobertura de Necessidades do CRM
Fonte: O Autor.

4.8 ANÁLISE DE INDEPENDÊNCIA E CONTEÚDO DE INFORMAÇÃO

Nesta seção é realizada a análise da independência funcional e a análise do conteúdo de informação (*i.e.,* análise da complexidade) para o *software* de CRM.

4.8.1 Análise sobre a Independência Funcional

A análise da independência funcional para estudo do *software* de CRM é importante para identificar os relacionamentos entre elementos (*i.e.,* CPs, CNs, FRs). A Figura 27 apresenta uma matriz de projeto (DM) com o relacionamento entre CPs e CNs. Nesta DM é possível identificar visualmente que se trata de uma matriz triangular, portanto aceitável no ponto de vista do AD. A principal análise a ser realizada sobre a Figura 27 diz respeito aos relacionamentos fora da diagonal principal (*e.g.,* CP.2 com CN.3, CN.3.1 e CN.3.2). O analista precisa isolar cada relacionamento e entender suas dependências com outros relacionamentos. Neste estudo de aplicação do método, as dependências funcionais são causadas pela natureza sequencial das tarefas de obtenção dos *feedbacks* dos clientes (CN.3) e a utilização desta informação em estatísticas (CN.2). A existência dos relacionamentos não implica uma dependência funcional neste caso, tornando, assim, a solução aceitável. Entretanto, se o analista desejasse uma solução ótima, ele poderia revisar a especificação dos problemas e/ou criar um novo grupo de necessidades (*i.e.,* a nova solução) e comparar as matrizes de projeto resultantes considerando a grau de independência funcional das novas soluções criadas.

	CN.1	CN.1.1	CN.1.2	CN.1.3	CN.2	CN.2.1	CN.2.2	CN.3	CN.4	CN.5
CP.1	X									
CP.1.1		X	X							
CP.1.2			X							
CP.1.3				X						
CP.2					X			X		
CP.2.1						X		X		
CP.2.2							X	X		
CP.3								X		
CP.4									X	
CP.5										X

Figura 27 - DM de Problemas e Necessidades do CRM
Fonte: O Autor.

Como próximo passo, o analista conduz a análise das dependências funcionais entre CNs e FRs. A Figura 28 apresenta a matriz de projeto para as necessidades e requisitos funcionais. Neste estudo identifica-se uma possível redundância para as CN.2.1 e CN.4, pois existe mais de um FR associado a uma única CN. Como previamente discutido, o analista precisa identificar se existe redundância nos FRs propostos ou se estes FRs são facetas diferentes da necessidade apresentada. Neste estudo, as decomposições dos FRs foram realizadas em um nível de detalhe maior que o requerido para as CNs.

	FR.1	FR.1.1	FR.1.2	FR.1.3	FR.2	FR.2.1	FR.2.1.1	FR.2.1.2	FR.2.1.3	FR.2.2	FR.3	FR.3.1	FR.3.2	FR.4	FR.4.1	FR.4.2	FR.4.3	FR.4.4	FR.4.5	FR.5
CN.1	X																			
CN.1.1		X																		
CN.1.2			X																	
CN.1.3				X																
CN.2					X															
CN.2.1						X	X	X	X											
CN.2.2										X										
CN.3											X	X	X							
CN.4														X	X	X	X	X	X	
CN.5																				X

Figura 28 - DM de Necessidades e Requisitos Funcionais
Fonte: O Autor.

Alternativamente, o analista poderia ajustar a especificação para formar uma matriz de projeto quadrada (como sugere o AD). Isso poderia ser feito suprimindo o nível adicional de decomposição do requisito FR.2.1 (*i.e.,* FR.2.1.1, FR.2.1.2, FR.2.1.3), de forma que FR.2.1 represente ou abstraia estes subrequisitos. Os FRs relacionados a CN.3 podem ser subrimidos da mesma forma que o FR.2.1. Além disso, para o caso da CN.4, pode-se criar dois novos subFRs (FR.4.1 e FR.4.2) e duas novas subCNs e abstrair 5 níveis anteriores (FR.4.1 à FR.4.5). A Figura 29 apresenta a nova DM com o resultado da análise.

	FR.1	FR.1.1	FR.1.2	FR.1.3	FR.2	FR.2.1	FR.2.2	FR.3	FR.4	FR.4.1	FR.4.2	FR.5
CN.1	X											
CN.1.1		X										
CN.1.2			X									
CN.1.3				X								
CN.2					X							
CN.2.1						X						
CN.2.2							X					
CN.3								X				
CN.4									X			
CN.4.1										X		
CN.4.2											X	
CN.5												X

Figura 29 - DM de Necessidades e Requisitos Funcionais Revisada
Fonte: O Autor.

Em resumo, a DM final para as CNs ilustrada na Figura 29 contém as seguintes modificações em comparação com a matriz inicial ilustrada na Figura 28:

- A hierarquia de terceiro nível N.N.N foram suprimidas dos FR.2.1 e FR.4 devido à não relevância na análise neste estudo.
- A hierarquia de segundo nível N.N foram suprimidas do FR.3 como forma de exemplo de estudo.
- Foi criado o segundo nível N.N de hierarquia para a relação CN.4 e FR.4.

O Quadro 24 e Quadro 25 apresentam os novos FRs e CNs criados após a análise de dependência entre CNs e FRs.

FR.4 - O *software* CRM deve permitir ao Gerente de Contas e ao Gerente de CRM gerenciar perfis de comportamento.
FR.4.1 - O *software* CRM deve permitir ao Gerente de CRM gerenciar perfis de clientes.
FR.4.2 - O *software* CRM deve permitir ao Gerente de CRM gerenciar os dados de comportamento dos clientes.

Quadro 24 - Especificação de Requisitos de *Software* Revisada para o FR.4
Fonte: O Autor.

CN.4 - O Gerente de Contas e ao Gerente de CRM objetivam um *software* CRM para ter conhecimento sobre o comportamento dos clientes. CN.4.1 - O Gerente de CRM quer um *software* CRM para saber sobre os perfis de clientes. CN.4.2 - O Gerente de Contas objetiva um *software* CRM para conhecer sobre o comportamento dos clientes de perfis específicos.

Quadro 25 - Especificação de Necessidades Revisada para a CN.4
Fonte: O Autor.

4.8.2 Cálculo do Conteúdo de Informação para Necessidades

A primeira tarefa do analista de *software* para calcular o conteúdo de informação de uma especificação é determinar a Faixa de Problema (*Problem Range*) para cada um dos CPs identificados. As Faixas de Problema para os CPs, apresentadas no Quadro 26, foram estabelecidas arbitrariamente baseadas em parâmetros dos problemas e pesos dos subproblemas.

Problemas	Faixas de Problema
CP.1 - A empresa deve assegurar a existência de um canal de comunicação com todos os clientes, caso contrário, perda de clientes pode ocorrer afetando marketing, promoções, feedbacks, e vendas futuras.	Acima de 70% de número de clientes com canal ativo. Estará satisfeito com qual meio de alcançar isso?
CP.1.1 - A empresa deve assegurar que pode contatar todos os clientes.	0,5
CP.1.2 - A empresa deve assegurar que cada cliente seja contatado regularmente.	0,3
CP.1.3 - A empresa deve assegurar um relacionamento ativo com cada cliente.	0,2
CP.2 - A empresa deve considerar as estatísticas sobre os feedbacks dos clientes no planejamento da empresa, caso contrário, acarretará na insatisfação dos clientes e consequentemente perda de vendas e fatia de mercado.	Acima de 15% de feedbacks sobre as vendas
CP.2.1 - A empresa tem que assegurar que os feedbacks dos clientes estão sendo considerados.	0,5
CP.2.2 - A empresa tem que assegurar que os cálculos estatísticos estão atualizados.	0,5
CP.3 - Clientes tem expectativa que a empresa responda seus feedbacks, caso contrário, os clientes se tornarão frustrados e a empresa pode sobre a penalidade de diminuição de reputação.	Acima de 70% de número de respostas aos clientes.
CP.4 - A empresa deve alinhar suas estratégias de venda e campanhas com o comportamento dos clientes, caso contrário, a empresa corre o risco de perder oportunidades de vendas e falhar em suas estratégias.	Acima de 50% do comportamento dos clientes é mapeado e considerado nas estratégias.
CP.4.1 - A empresa tem que assegurar que os diferentes perfis dos clientes estão sendo considerados.	0,5
CP.4.2 - A empresa tem que assegurar que o comportamento dos clientes está sendo considerados.	0,5
CP.5 - A empresa deve projetar as vendas caso contrário pode perder oportunidades e fazer provisões erradas.	Projeções de vendas (100%) atualizadas mensalmente.

Quadro 26 - Faixas de Problema
Fonte: O Autor.

Para o cálculo do conteúdo da informação das CNs foi utilizado a Equação 1 e as Faixas de Problema e Necessidade. Para as CN.1 e CN.2 foram utilizados a soma do conteúdo da informação (determinado pela Equação 1) das subnecessidades (SUH, 2005) como exemplificado pela Equação 4 para a CN.1.

$$\mathbf{0 + \ln(0{,}5 / 0{,}3) = 0{,}510825624} \quad (4)$$

A Tabela 4 apresenta o conteúdo da informação para as necessidades calculado utilizando a Equação 1, o grau de satisfação e a Faixa de Necessidade, ou *Need Range*, que foi estabelecido de forma arbitrária como uma função de satisfação. Neste estudo, as necessidades CN.3 e CN.5 têm o conteúdo de informação igual à zero, o que significa que satisfazem completamente os problemas relacionados. Por outro lado, as necessidades CN.1, CN.2 e CN.4 apresentam valores numéricos diferente de zero relativo à satisfação dos problemas relacionados. Isso se deve ao fato de que se julgou que estas necessidades não satisfazem completamente os problemas relacionados. Neste caso, o analista pode procurar soluções alternativas de necessidades para aumentar o índice de satisfação. O conteúdo da informação para todas as CNs está representado na coluna "Conteúdo da Informação Total".

Tabela 4 - Conteúdo da Informação para CNs do CRM

Necessidades	Faixa de Problema	Faixa de Necessidade	Conteúdo da Informação para subCNs	Conteúdo da Informação para CNs	Conteúdo da Informação Total
CN.1				0,510825624	0,510825624
CN.1.1	0,5	1	0		
CN.1.2	0,3	0,5	0,510825624		
CN.2				0	
CN.2.1	0,5	1	0		
CN.2.2	0,5	1	0		
CN.3	1	1		0	
CN.4	0,5	1		0	
CN.5	1	1		0	

Fonte: O Autor.

4.8.3 Cálculo do Conteúdo de Informação para Requisitos Funcionais

O próximo passo na aplicação do AD para o estudo do sistema de CRM é o cálculo do conteúdo da informação para os requisitos funcionais (FR). Neste caso, tem-se quatro FRs que têm subníveis e o FR.5 sem decomposição, como apresentado na Tabela 5. O conteúdo da informação é calculado neste estudo para o nível mais alto da hierarquia a partir da avaliação do grau de satisfação dos subFRs. Desta forma, a Faixa de Necessidade é estabelecida utilizando pesos para cada subCN que, somados, resultam no valor 1 (SUH, 1990, p. 154). A Faixa de Necessidade, nesta análise, representa o *design range*. A Faixa de Requisito, que representa o *system range,* foi definida como a função de satisfação de cada um dos FRs. A coluna "Faixa de Requisito" na Tabela 5, indica a avaliação feita pelo analista sobre o quanto cada FR satisfaz a

necessidade associada (conforme relacionamentos indicados no Figura 29). Utilizando uma soma de todos os conteúdos de informação (determinado pela Equação 1) dos subFRs, é possível determinar no conteúdo da informação para cada um dos FR de alto-nível, conforme apresentado na coluna “Conteúdo de Informação para FRs” na Tabela 5.

O conteúdo de informação de toda a especificação da solução de CRM é dado pela soma dos conteúdos de informação associado a cada FR (SUH, 2005), totalizando 7,436599408 neste estudo (conforme a coluna a direita na Tabela 5). Como este valor é maior que zero, há a possibilidade de reavaliar a especificação visando minimizá-lo. De acordo com o AD, entretanto, o conteúdo de informação é uma medida de comparação entre soluções alternativas. Assim, se houver ou for construída uma segunda solução de especificação para o mesmo sistema, pode-se usar esta medida para compará-las.

Tabela 5 - Conteúdo da Informação para FRs

Requisitos Funcionais	Faixa de Necessidade	Faixa de Requisito	Conteúdo da Informação para subFRs	Conteúdo da Informação para FRs	Conteúdo da Informação Total
FR.1				0,751987681	7,436599408
FR.1.1	0,33	1	0		
FR.1.2	0,33	0,7	0,751987681		
FR.1.3	0,33	1	0		
FR.2				0,916290732	
FR.2.1	0,5	1	0		
FR.2.2	0,5	0,2	0,916290732		
FR.3				2,079441542	
FR.3.1	0,5	0,125	1,386294361		
FR.3.2	0,5	0,25	0,693147181		
FR.4				3,688879454	
FR.4.1	0,2	1	0		
FR.4.2	0,2	1	0		
FR.4.3	0,2	1	0		
FR.4.4	0,2	0,02	2,302585093		
FR.4.5	0,2	0,05	1,386294361		
FR.5	1	1		0	

Fonte: O Autor.

4.9 DISCUSSÃO SOBRE O CASO DE ESTUDO

O estudo apresentado nesta seção teve como propósito ilustrar a aplicação do método *Problem-Based SRS* por meio da especificação de requisitos de *software* no contexto de um *software* CRM. A utilização do método

Problem-Based SRS direcionou a compreensão sobre a especificação do *software* e definiu uma sequência lógica para construção desta SRS. Adicionalmente, o método auxiliou o analista a revisar se as informações especificadas neste estudo atendem as questões de qualidade da especificação de requisitos.

Como resultado, a SRS para o *software* de CRM conta com a descrição clara do contexto, problemas, necessidades do cliente, requisitos e, também, de aspectos de *design* como o *software glance* e visão de *software*. Esta forma de especificação foi essencial para o analista compreender e estabelecer um acordo com os *stakeholders* sobre a justificativa de construção de uma solução de *software*, juntamente seu desenho (*design*) preliminar na forma de seu *software glance*. A especificação das necessidades, que foi orientada à resolução dos problemas, esclarece os principais resultados a serem entregues pelo *software* e prepararam para o detalhamento da solução na forma da visão do *software*. A utilização da especificação de necessidades e do *software glance* para a construção da visão do *software* auxilia o analista na identificação de diversos aspectos do *software* como as principais funcionalidades, o ambiente no qual o *software* será utilizado e os principais *stakeholders* que irão interagir com o *software*. O último estágio de detalhe da SRS é a especificação dos requisitos funcionais que estabelecem, com uma riqueza maior de detalhe, o que o *software* deverá fazer (ou prover como funções).

Utilizando o *Problem-Based SRS,* os requisitos funcionais especificados têm um embasamento e justificativa muito mais consistentes se comparados com uma lista convencional de requisitos. Neste estudo do CRM, ilustrou-se, também, como analisar aspectos de qualidade dos requisitos de *software* na forma de rastreabilidade de problemas e necessidades que auxiliam o analista a identificar possíveis lacunas ou não cobertura na solução. A análise é uma parte da técnica de especificação de requisitos e atende as normas de qualidade de requisitos de *software*.

O objeto do estudo (sistema CRM) possui pequena extensão, de forma que ele permitiu ilustrar apenas os principais aspectos da aplicação do método proposto. A forma da especificação deste sistema derivou das decisões tomadas pelo autor na posição de “analista” do estudo. Diferentes analistas, poderiam

construir especificações distintas para o mesmo contexto de negócio, estabelecendo diferentes problemas, necessidades e requisitos. Isto levaria a especificações distintas, entre as quais, algumas seriam mais adequadas que outras para as intenções originais dos clientes. O método proposto, se usado na criação de destas alternativas de especificação, colaboraria para garantir que todas possuem uma consistência com relação ao contexto de negócio. Ainda assim, usando o AD como base para análise, algumas especificações teriam maior qualidade que outras e poderiam ser diferenciadas para a escolha da melhor solução.

5 EXPERIMENTO COM O MÉTODO *PROBLEM-BASED SRS*

Em adição ao estudo da aplicação do método apresentado no capítulo anterior, desenvolveu-se um experimento sobre um projeto real, denominado, MicroER. Este projeto visa o desenvolvimento de um sistema computacional para gerenciamento da produção e consumo de energia elétrica renovável que está sendo desenvolvido pela UTFPR no Laboratório de Inovações Tecnológicas (LIT) / Centro de Inovações Tecnológicas (CITEC) em parceria com a National Instruments (NI) e com apoio financeiro solicitado ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

O experimento foi dividido com base nos processos do método *Problem-Based SRS*. A subseção de "contexto de negócio" apresenta uma visão geral do sistema MicroER e as características vislumbradas para o sistema. A subseção de "problemas do cliente" detalha os problemas construídos com base no contexto e detalhes fornecidos pelos *stakeholders*. A subseção de "*software glance"* contém o início do *design* da solução de *software* com aspectos de alto nível com vistas à resolução dos problemas apresentados. A subseção de "necessidades dos clientes" detalha os aspectos que o *software* terá que comportar seguindo o *software glance* para ajudar a resolver os problemas. A subseção de "visão do *software"* apresenta o *design* ampliado do *software glance* contendo aspectos como ambiente e lista de *stakeholders*. A subseção de "requisitos de *software"* detalha o nível mais baixo de abstração da SRS contendo aspectos do domínio funcional da solução. Por fim, apresenta-se uma discussão sobre o experimento realizado.

5.1 CONTEXTO DE NEGÓCIO

A geração de energia renovável alcançou as residências das pessoas por meio de unidades de microgeração de energia renovável com preços cada vez mais acessíveis. Entretanto, os sistemas atuais ainda apresentam várias limitações e problemas que necessitam ser resolvidos para maximizar a eficiência e flexibilizar seu uso. O projeto MicroER da UTFPR (STADZISZ, *et al.*, 2016) tem o objetivo de desenvolver um sistema eletrônico de gerenciamento e otimização relacionados de uma unidade de microgeração contendo fontes de energia solar e eólica. Utilizando componentes de *hardware* e *software,* o

sistema MicroER auxiliará os usuários finais a ajustar dinamicamente seu consumo de energia de acordo com suas preferências (perfis de consumo). Consequentemente, o sistema pode diminuir o consumo e estender a duração de uso de energia gerada por fontes renováveis (STADZISZ, *et al.*, 2016).

A Figura 30 ilustra uma visão arquitetural dos módulos eletrônicos de gerenciamento que compõem o sistema MicroER (parte superior da figura) e dos módulos que compõem a unidade de microgeração (parte inferior da figura).

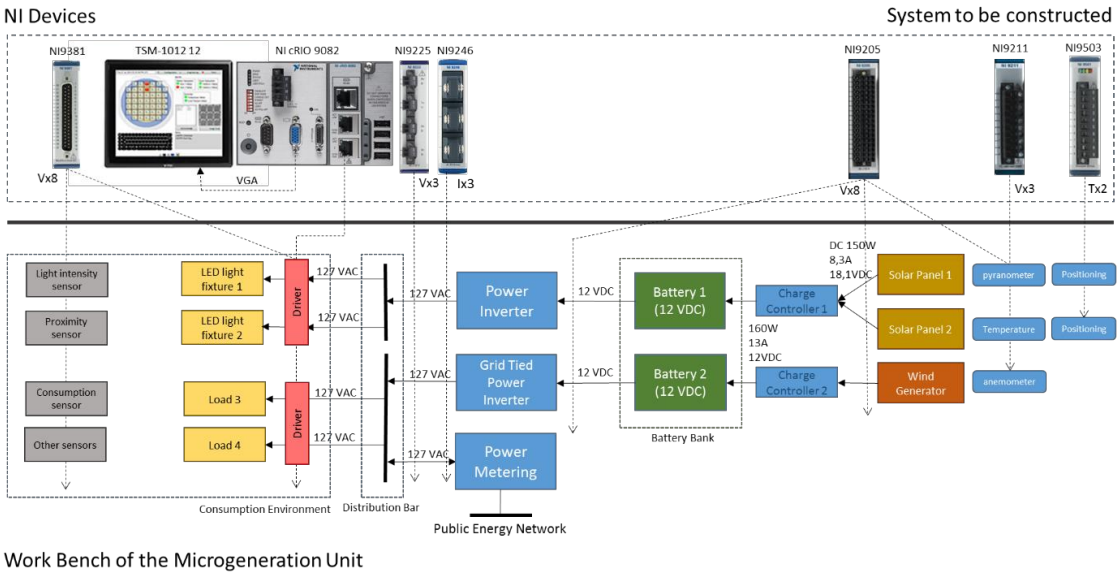


Figura 30 - Esquema do Projeto MicroER
Fonte: Stadzisz, *et.al* (2016)

O Quadro 27 apresenta o contexto de negócio definido para aplicação do método *Problem-Based SRS* neste experimento. O contexto do cliente, entretanto, pode ser muito mais amplo do que uma descrição breve em linguagem natural. Assim, sua descrição pode se estender a: diagramas como a Figura 30 contendo o esquema do sistema no qual o MicroER irá ser implementado, resultados de entrevistas com os usuários do sistema e de análises de documentos relacionados, além de regulamentações e modelos de negócio.

Contexto de Negócio:

O usuário final residencial preocupa-se com o uso racional das fontes de energia e escolheu investir na produção de energia renovável por meio de uma unidade microgeradora de energia com fontes solar e eólica. O usuário

também tem interesse em reduzir o seu consumo, como meio para diminuir impacto na matriz energética regional e nacional. O usuário deverá ser capaz de interagir com os mecanismos de controle de forma a poder adequá-los a seu perfil de consumo e de evoluir este perfil de acordo com sua experiência no uso de energias renováveis. O usuário poderá ter conhecimento da eficiência do sistema, de sua qualidade de operação e de como seu perfil de consumo se adequa às condições energéticas providas pelo sistema de microgeração.

Quadro 27 - Contexto de Negócio para o MicroER
Fonte: O Autor adaptado de (STADZISZ, *et al.*, 2016).

5.2 PROBLEMAS DO CLIENTE

Utilizando o método *Problem-Based SRS,* o analista inicia a especificação do sistema pela definição dos CPs. Nesta etapa, o analista deve realizar a análise do contexto do cliente com objetivo de entender os deveres, intenções e anseios, além dos efeitos ou consequências, caso não seja possível atender estas demandas.

O intuito do processo de Especificação de Problemas do Cliente no método *Problem-Based SRS* é definir os problemas a serem resolvidos e apresentá-los de forma consistente. Os problemas podem estar em diferentes graus de abstração e podem ser decompostos em níveis criando, assim, subproblemas. O processo de decomposição pode ocorrer nesta fase de especificação de problemas ou em outras fases do método, isto ocorre devido à evolução da especificação nos outros domínios em uma abordagem iterativa (ou *zigzagging*). O analista pode seguir o processo a partir de apenas um problema, caso julgue apropriado e, então, em iterações posteriores, outros problemas podem ser definidos e detalhados. Para este experimento do MicroER, todos os problemas foram especificados conjuntamente, conforme apresentado no Quadro 28.

CP.1 - O cliente intenciona reduzir seu consumo visando diminuir o custo com energia elétrica fazendo uso de um sistema de microgeração de energia renovável baseado em fonte solar e eólica.

CP.1.1 - O cliente deve reduzir o consumo desnecessário de energia para que possa reduzir o custo com energia.
CP.1.2 - O cliente deve mudar seu padrão de consumo, sob pena de não conseguir reduzir seu consumo.
CP.1.3 - O cliente deve acompanhar seu padrão de consumo, sob pena de não conseguir reduzir seu consumo.
CP.1.4 - O cliente deve garantir a eficiência da unidade geradora, sob pena de não obter a energia necessária que a unidade pode gerar e ter que consumir da rede pública.
CP.1.5 - O cliente deve garantir o estado de manutenção da unidade geradora, sob pena de falha de funcionamento e ter que consumir energia da rede pública.
CP.2 - O cliente pretende contribuir para minimizar os impactos ambientais das fontes públicas de energia.
CP.2.1 - O cliente deve ter ciência do impacto ambiental de seu consumo.
CP.2.2 - O cliente intenciona racionalizar o uso da energia.
CP.3 - O cliente intenciona contribuir para a redução da pressão de consumo sobre a matriz energética regional e nacional.
CP.4 - O cliente intenciona influenciar outros indivíduos para a causa energética e ambiental.

Quadro 28 - Problemas do Cliente para o MicroER
Fonte: O Autor.

O problema CP.1 especifica a intenção do cliente em reduzir seu consumo energético. Este problema contém três subproblemas relacionados ao aspecto da redução do consumo. A primeira faceta ou subproblema CP.1.1 está relacionado ao modo de uso da energia dentro da residência, especificamente o desperdício gerado (*e.g.,* lâmpadas acessas sem pessoas no local). O CP.1.2 especifica aspectos relacionados ao comportamento de uso com a preocupação sobre padrões de consumo. O CP.1.3 especifica que o cliente deve acompanhar ou monitorar seu consumo para adequação ao longo do uso. O CP.1.4 especifica o aspecto de acompanhamento da eficiência dos equipamentos ou do sistema.

O CP.1.5 está relacionado a aspectos de manutenção e, também como o CP.1.4, gera uma penalidade caso não seja atendido.

O problema CP.2, especifica como o cliente pretende contribuir para minimizar os impactos ambientas e foi particionado em dois subCPs. O primeiro subCP é relacionado ao entendimento e consciência (CP.2.1) sobre o seu consumo das fontes energéticas e o segundo subCP, é relacionado a racionalizar o consumo (CP.2.2). CP.3 e CP.4 não necessitaram particionamento.

5.3 SOFTWARE GLANCE

O *software glance* trata da primeira representação da ideia de organização do sistema MicroER que está sendo vislumbrada pelo analista após conhecer os problemas do cliente. No experimento, o analista propôs uma solução de *software* que utiliza integrações com componentes específicos de *hardware* que recebem comandos e produzem informações. O Quadro 29 apresenta o *software glance* para o MicroER que tem o objetivo de ajudar a resolver os problemas apresentados no Quadro 28.

O *software* do sistema MicroER tem como principal objetivo gerenciar equipamentos e sensores integrados à uma unidade de microgeração de energia na residência do usuário final. Este *software* de gerenciamento deve realizar a aquisição de sinais de diferentes equipamentos e sensores e armazená-los em uma base de dados. Através de uma interface ser humano-computador (IHM), o usuário pode acompanhar seu padrão de consumo, qualidade de operação do sistema e registrar perfis de consumo. Adicionalmente, ele pode ativar ou desativar equipamentos através de componentes de comando.

Quadro 29 - *Software Glance* para o MicroER

A Figura 31 ilustra graficamente o *software glance* descrito no Quadro 29 utilizando um diagrama de blocos. Neste diagrama, o sistema MicroER é dividido em dois subsistemas: o *software* e o ambiente ou residência do cliente. O subsistema de *software* apresenta os componentes de controle, aquisição e comando que persistem e consultam informações em uma base de dados. O

subsistema de Ambiente envolve os equipamentos, sensores e interface presente na residência do usuário final.

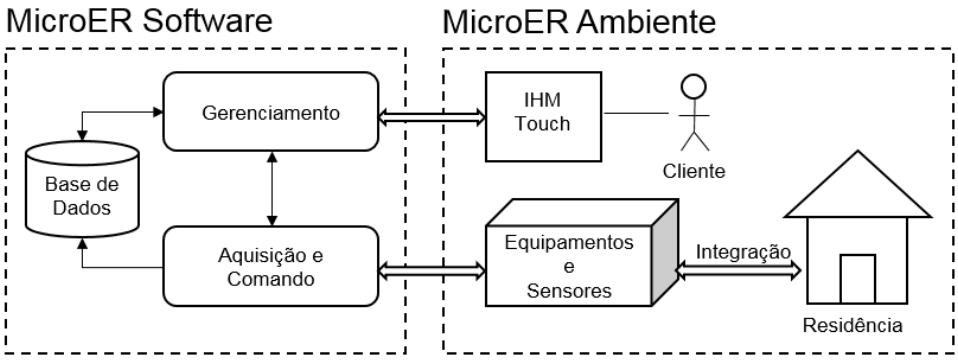


Figura 31 - Diagrama *Software Glance do MicroER*
Fonte: O Autor.

5.4 NECESSIDADES DO CLIENTE

A especificação das necessidades do cliente para o experimento do MicroER tem como objetivo especificar soluções de *software* que: (*i*) apresentem informações para os clientes e (*ii*) controlem parâmetros do sistema ajudem que o cliente a resolver os problemas apresentados. O Quadro 30 apresenta as necessidades especificadas.

CN.1 - O cliente precisa de um *software* para monitorar e controlar a produção e o consumo energético.
CN.1.1 - O cliente precisa de um *software* para ter conhecimento de seu padrão de consumo (quanto e com o que consume energia elétrica).
CN.1.2 - O cliente necessita de um *software* para ajustar o consumo em função da capacidade energética renovável e do perfil de consumo.
CN.1.3 - O cliente deseja um *software* para ter conhecimento da produção energética da unidade geradora.
CN.1.4 - O cliente precisa de um *software* para ser alertado sobre a queda de eficiência da geração de energia.
CN.1.5 - O cliente precisa de um *software* para ser alertado sobre as exigências de manutenção da unidade geradora.

CN.2 - O cliente precisa de um *software* para ter conhecimento da racionalização de seu consumo.

CN.2.1 - O cliente quer um *software* para saber o quanto seu consumo está racionalizado e colaborando com os fatores ambientais.

CN.2.2 - O cliente deseja um *software* para acompanhar a evolução da racionalização de seu padrão de consumo.

CN.3 - O cliente quer saber a relação de independência de seu consumo da rede elétrica pública.

CN.4 - O cliente quer um *software* para saber os benefícios gerados com o melhor aproveitamento da fonte energética para divulgar a outras pessoas.

Quadro 30 - Necessidades do Cliente para MicroER
Fonte: O Autor.

5.5 VISÃO DO *SOFTWARE*

A especificação da visão do *software* para o MicroER utiliza o *software glance*, apresentado na Figura 31 e especificado no Quadro 29, juntamente com as necessidades no Quadro 30 como ponto de início na proposta de uma solução mais completa em aspectos como funcionalidades de alto-nível, ambiente, *stakeholders* e posicionamento. O Quadro 31 apresenta esta visão do *software*.

Posicionamento: o MicroER é um *software* que auxilia o cliente final residencial a racionalizar o consumo elétrico por meio de uma plataforma inteligente que integra-se com um sistema de microgeração de energia renovável baseado em fonte solar e eólica tendo impacto em menor custo energético. O cliente poderá monitorar seu consumo e alterar aspectos de utilização de energia através de uma interface gráfica que, de forma intuitiva, auxilia a tomada de decisão sobre qual é o melhor perfil de consumo para o momento e beneficiando, assim, positivamente o meio ambiente.

Stakeholders: o Cliente final residencial

Funcionalidades de Alto-Nível:

- O *software* MicroER deve garantir o melhor aproveitamento do sistema de geração de energia renovável.

- O *software* MicroER deve fornecer informações sobre o padrão de consumo, eficiência e impacto ambiental.
- O *software* MicroER deve fornecer uma interface gráfica para gerenciamento de perfis de consumo.

Ambiente: o *software* MicroER proporciona uma integração total com a residência do cliente. Através de uma interface gráfica, o cliente pode de forma intuitiva modificar o sistema para máxima eficiência na utilização de recursos renováveis. Mecanismos de gerenciamento auxiliam na integração com componentes instalados na residência como geradores, lâmpadas e tomadas.

Quadro 31 - Visão do *Software* para MicroER
Fonte: O Autor.

A Figura 32 ilustra, por meio de um diagrama de blocos, a visão de *software* para o MicroER. Nesta representação é possível ver um nível maior de detalhe comparado ao *software glance* ilustrado na Figura 31. Interfaces estão melhor definidas. Por exemplo, para o ajuste de consumo será utilizado a interface de comunicação serial RS485 por meio da qual o componente de *software* pode estabelecer um protocolo de comunicação com os *drivers*. O componente aquisição e controle também sofreu alteração comparado com o *software glance,* decompondo-se em Acionamento, Aquisição de Sinais e Controle de IHM. Equipamentos e Sensores vislumbrados tornaram-se *drivers*, acionadores e unidades de aquisição na visão de *software*.

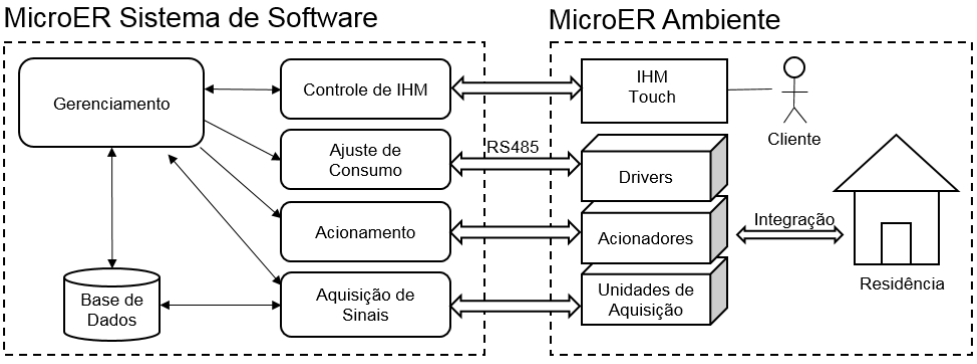


Figura 32 - Diagrama Visão do *Software* para MicroER
Fonte: O Autor.

5.6 REQUISITOS DE *SOFTWARE*

A etapa de especificação de requisitos de *software* produz o maior nível de detalhe no método *Problem-Based SRS*. Cada requisito é produzido através do processo de especificação que mapeia cada necessidade do Quadro 30 e cria um requisito no mesmo nível de abstração que responda a esta necessidade e esteja dentro da visão do *software* descrita no Quadro 3. O analista de *software* pode utilizar a técnica de *zigzagging* para decompor níveis mais baixos dos requisitos funcionais. O Quadro 32 apresenta os FRs produzidos com base na visão de *software* e necessidades apresentadas.

Adicionalmente, nesta subseção é possível observar a lista de Requisitos Não-Funcionais (NFRs) que representam aspectos técnicos ou restrições.

FR.1. - O *software* MicroER deve controlar e informar ao cliente todos os aspectos da produção e consumo energético da residência.

FR.1.1 - O *software* MicroER deve possibilitar ao Cliente visualizar o consumo de energia elétrica por equipamento dividido por fonte energética.

FR.1.2 - O *software* MicroER deve utilizar e possibilitar o gerenciamento de perfis de consumo.

FR.1.2.1 - O *software* MicroER deve possibilitar ao Cliente registrar perfis de consumo.

FR.1.2.2 - O *software* MicroER deve possibilitar ao Cliente modificar perfis de consumo.

FR.1.2.3 - O *software* MicroER deve ajustar os parâmetros dos equipamentos de acordo com o Perfil do Cliente.

FR.1.2.4 - O *software* MicroER deve ajustar os parâmetros dos equipamentos de acordo disponibilidade energética e perfil do Cliente.

FR.1.3 - O *software* MicroER deve informar sobre a produção energética da residência.

FR.1.3.1- O *software* MicroER deve calcular a capacidade energética disponível com base nas informações das unidades geradoras e armazenar na base de dados.

FR.1.3.2 - O *software* MicroER deve comparar os parâmetros armazenados com as informações do momento das unidades geradoras e armazenar o resultado na base de dados.

FR.1.4 - O *software* MicroER deve possibilitar ao Cliente visualizar a eficiência e estado das unidades geradoras.

FR.1.5 - O *software* MicroER deve possibilitar ao Cliente visualizar os períodos de manutenção das unidades geradoras.

FR.2 - O *software* MicroER deve apresentar dados sobre a racionalização do consumo da residência.

FR.2.1 - O *software* MicroER deve apresentar dados relacionados a fatores ambientais.

FR.2.1.1 - O *software* MicroER deve calcular fatores ambientais e armazená-los na base de dados a cada 5 segundos.

FR.2.1.2 - O *software* MicroER deve possibilitar ao Cliente visualizar fatores ambientais disponíveis.

FR.2.2 - O *software* MicroER deve racionalizar a utilização de energia baseado em projeções de consumo e perfis de uso.

FR.2.2.1 - O *software* MicroER deve projetar a capacidade energética tendo em consideração o perfil energético do Cliente e a disponibilidade energética atual e armazenar na base de dados.

FR.2.2.2 - O *software* MicroER deve ajustar os parâmetros dos equipamentos de acordo com a projeção de consumo.

FR.3 - O *software* MicroER deve informar sobre a independência energética da residência.

FR.4 - O *software* MicroER deve permitir ao Cliente visualizar todos os benefícios de economia gerados.

NFR.1 - O *software* MicroER deve apresentar ao Cliente uma interface gráfica ao usuário em um *display* de 10 polegadas. NFR.2 - Os parâmetros de qualidade de energia serão coletados a cada 5 segundos (FR.1.3.2).

Quadro 32 - Especificação de Requisitos de *Software* para MicroER
Fonte: O Autor.

5.7 ANÁLISE DE RASTREABILIDADE

A análise de rastreabilidade é utilizada pelo analista de *software* com o objetivo de descobrir se todos os CPs e CNs foram atendidos. Para isto esta análise é dividida em dois passos. A Figura 33 apresenta a matriz de rastreabilidade para análise de cobertura e conformidade de problemas. A Figura 34 apresenta a RTM para análise de rastreabilidade e conformidade de necessidades.

Para o experimento do MicroER tem-se dois tipos de grupos de CPs e CNs. (*i*) um CP com uma única CN. (*ii*) CPs decompostos com CNs decompostas. Nestas relações é possível observar que o grupo *i* é classificado como completo (C) ou atende integralmente a relação. O grupo *ii* apresenta uma hierarquia de decomposição, no qual, seus sub elementos satisfazem parcialmente (P) se analisados individualmente o problema ou necessidade de nível mais superior. Neste segundo grupo quando o analista de *software* analisa todos os elementos pode considerar o resultado como completo (C) no nível mais macro destes CPs ou CNs.

	CN.1	CN1.1	CN.1.2	CN1.3	CN1.4	CN1.5	CN.2	CN.2.1	CN.2.2	CN.3	CN.4
CP1	C										
CP.1.1		P									
CP.1.2			P								
CP.1.3				P							
CP.1.4					P						
CP.1.5						P					
CP.2							C				
CP.2.1								P			
CP.2.2									P		
CP.3										C	
CP.4											C

Figura 33 - RTM de Análise de Cobertura de Problemas
Fonte: O Autor.

A Figura 34 apresenta a matriz de rastreabilidade para análise de cobertura de necessidades. A análise de conformidade para problemas e

necessidades tem o mesmo objetivo que é encontrar elementos não associados ou associações desnecessárias ou redundantes. Em ambos os casos de RTM do experimento do MicroER, as CNs e FRs resultantes são necessários (não há colunas vazias) e suas associações não apresentam redundância (não há mais de uma solução (C ou P) para cada CP ou CN.

	FR.1	FR.1.1	FR.1.2	FR.1.3	FR.1.4	FR.1.5	FR.2	FR.2.1	FR.2.2	FR.3	FR.4
CN.1	C										
CN.1.1		P									
CN.1.2			P								
CN.1.3				P							
CN.1.4					P						
CN.1.5						P					
CN.2							C				
CN.2.1								P			
CN.2.2									P		
CN.3										C	
CN.4											C

Figura 34 - RTM de Análise de Cobertura de Necessidades
Fonte: O Autor.

5.8 ANÁLISE DE INDEPENDÊNCIA E CONTEÚDO DE INFORMAÇÂO

Nesta subseção é apresentado a aplicação dos princípios do *axiomatic design,* que se resumem a aplicação do Axioma 1 e 2. Outra utilizadade do AD é a comparação entre propostas de projeto, porém para o experimento do MicroER apenas uma proposta de projetos foi desenvolvida.

A análise da independência no método *Problem-Based SRS* corresponde à aplicação do Axioma 1 nos domínios de problemas e necessidades. Ela tem como objetivo identificar se os elementos (*i.g.,* CPs ou CNs) existentes sofrem ou exercem interferência funcional em outros elementos dentro da matriz de projeto. Esta interferência deve ser resolvida por meio da aplicação dos princípios do AD. Esta interferência pode ser causada, por exemplo, devido a uma associação desnecessária com um outro elemento. A Figura 35 ilustra a matriz de projeto para análise de independência entre CPs e CNs para o caso MicroER.

	CN.1	CN1.1	CN.1.2	CN1.3	CN1.4	CN1.5	CN.2	CN.2.1	CN.2.2	CN.3	CN.4
CP1	X										
CP.1.1		X									
CP.1.2			X								
CP.1.3				X							
CP.1.4					X						
CP.1.5						X					
CP.2							X				
CP.2.1								X			
CP.2.2									X		
CP.3										X	
CP.4											X

Figura 35 - DM de Análise de Independência de CPs e CNs
Fonte: O Autor.

No caso ilustrado na Figura 35, a solução desenvolvida pelo analista de *software* produz uma matriz de projeto não acomplada (ou ideal) de acordo com o teorema 4. Nesta DM, o resultado ilustrado graficamente é uma diagonal principal de associações entre CP e CNs sem interferências funcionais. Devido ao entendimento prévio sobre os princípios do AD, o analista de *software* tende a sintetizar uma solução que não viole o axioma 1, o que quer dizer, que não contenha interferência funcional entre os elementos.

O próximo nível de análise é a relação entre FRs e CNs. A Figura 36 ilustra uma DM quadrada, ou seja, que possui a mesma quantidade de elementos nos eixos CN e FR e a relação entre os elementos não apresenta interferências funcionais.

	FR.1	FR.1.1	FR.1.2	FR.1.3	FR.1.4	FR.1.5	FR.2	FR.2.1	FR.2.2	FR.3	FR.4
CN.1	X										
CN.1.1		X									
CN.1.2			X								
CN.1.3				X							
CN.1.4					X						
CN.1.5						X					
CN.2							X				
CN.2.1								X			
CN.2.2									X		
CN.3										X	
CN.4											X

Figura 36 - DM de Análise de Independência de CNs e FRs
Fonte: O Autor.

A segunda etapa da análise com os princípios do AD é o cálculo do conteúdo da informação. Este cálculo do conteúdo é dividido em duas etapas: a primeira para as necessidades e a segunda para os requisitos.

Inicialmente, foi definida a Faixa de Problema ilustrada na Figura 37 com pesos para cada uma dos subproblemas (*i.g.,* CP.N.N) que somados resultam no valor 1 que é o peso do problema de alto-nível (SUH, 1990).

Para o cálculo do conteúdo da informação das CNs, o analisa de *software* precisa identificar informações como o nível de satisfação do cliente com cada uma das necessidades apresentadas e, caso estas necessidades estejam decompostas em subnecessidades, qual é o grau de satisfação para cada uma destas subnecessidades. A Faixa de Necessidade está ilustrada na Figura 37 e inclui os níveis de satisfação avaliados pelo analista para cada uma das CNs.

A combinação das Faixas de Problema e Necessidade utilizando a Equação 1 fornece a definição do conteúdo da informação para cada uma das CNs (conforme tabela da direita na Figura 37). Para as CN.1 e CN.2 foi necessário calcular o conteúdo da informação para as subCNs utilizando a Equação 1 e então determinar o conteúdo da informação das CNs de alto nível.

A Figura 37 apresenta a faixa de problema, a faixa de necessidade e o resultado do cálculo do conteúdo da informação para as CNs.

Problemas	Faixa de Problema
CP.1	
CP.1.1	0,2
CP.1.2	0,2
CP.1.3	0,2
CP.1.4	0,2
CP.1.5	0,2
CP.2	
CP.2.1	0,5
CP.2.2	0,5
CP.3	1
CP.4	1

Necessidades	Faixa de Necessidade	Conteúdo da Informação para as subCNs	Conteúdo da Informação para as CNs	Conteúdo da Informação Total
CN.1				
CN.1.1	1	0	2,505525937	3,786459782
CN.1.2	1	0		
CN.1.3	1	0		
CN.1.4	0,7	1,252762968		
CN.1.5	0,7	1,252762968		
CN.2				
CN.2.1	1	0	0,587786665	
CN.2.2	0,9	0,587786665		
CN.3	0,5		0,693147181	
CN.4	1		0	

Figura 37 - Faixas de Problema, Necessidade e CI para CNs - MicroER
Fonte: O Autor.

O resultado apresentado pelo cálculo do conteúdo da informação é utilizado como parâmetro quantitativo de qualidade de uma solução e pode ser utilizado para comparar projetos diferentes. É importante observar que algumas necessidades (*i.g.,* CN.1.4, CN.1.5, CN.2.2, CN.3) não satisfazem completamente o cliente e poderiam ser melhoradas, o que significa, que poderiam ser propostas novas soluções com a tentativa de diminuir o conteúdo

da informação até o valor zero, que é traduzido como de menor conteúdo ou menor complexidade.

A Tabela 6 apresenta o cálculo do conteúdo da informação para FRs. Para o cálculo das subFRs foi utilizado a Equação 1 e para o conteúdo da informação da FR de alto nível é utilizado a soma das subFRs (SUH, 2005). O conteúdo de informação de toda a especificação da solução do MicroER é dado pela soma dos conteúdos de informação associado a cada FR (SUH, 2005), totalizando 2,772588722 neste experimento (conforme a coluna a direita na Tabela 6).

Tabela 6 - Conteúdo da Informação para os FRs - MicroER

Requisitos Funcionais	Faixa de Necessidade	Faixa de Requisitos	Conteúdo da Informação para subFRs	Conteúdo da Informação para o FR	Conteúdo da Informação Total
FR.1					
FR.1.1	0,2	0,8	1,386294361		
FR.1.2	0,2	1	0	2,772588722	
FR.1.3	0,2	1	0		
FR.1.4	0,2	0,8	1,386294361		
FR.1.5	0,2	1	0		2,772588722
FR.2					
FR.2.1	0,5	0,5	0	0	
FR.2.2	0,5	1	0		
FR.3	1	1		0	
FR.4	1	1		0	

Fonte: O Autor.

5.9 DISCUSSÕES SOBRE O EXPERIMENTO

O experimento apresentado neste capítulo representa um caso real de projeto de um sistema de gerenciamento de geração e consumo de energia renovável em desenvolvimento na UTFPR. O objetivo deste experimento foi demonstrar a aplicação do método em uma situação realística, em complemento ao estudo apresentado no Capítulo 4 que envolveu um caso fictício.

O número de problemas, necessidades e requisitos especificados foi semelhante ao do estudo da aplicação do método discutido no Capítulo 4. As relações entre CPs, CNs e FRs foram, porém, mais “simples”, no sentido de haverem menor número de associações (e dependências). As RTMs de rastreabilidade indicaram que todos os problemas e necessidades foram cobertos pelos requisitos e que todos os requisitos e necessidades são justificáveis.

No tocante à aplicação da análise de independência, o resultado obtido nas matrizes de projeto de CPs x CNs e CNs x FRs indicou uma especificação “ideal”, como ilustrado na Figura 35 e Figura 36. Este resultado é esperado quando o analista de *software* aplica os princípios do AD desde das fases iniciais do projeto. No momento do mapeamento entre os domínios é natural para o analista aplicar o processo *zigzagging,* tornando, desta forma, os CPs, CNs e FRs independentes. Adicionalmente, o método proposto prevê a utilização de técnicas de rastreabilidade que mitigam erros comuns, preparando assim as informações para a avaliação, pelo ponto de vista, do AD. A utilização da matriz de projeto traz um ganho de qualidade significativo durante a fase de especificação dos domínios mapeados (*i.e., zig* mapeando CNs e FRs) e, então, decompostos (*i.g., zag* decompondo subCPs, subCNs).

Outro fator relacionado à independência, é o fato de que o analista pode realizar a síntese e análise individualmente por CP. Assim, o analista pode tratar aquele problema de forma independente em outros domínios subsequentes. A DM completa com todos os CPs auxilia na identificação de possíveis interferências pontuais que devem ser analisadas do ponto de vista de dependência funcional e minimizadas ou removidas, seguindo os princípios do AD.

6 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as considerações finais deste trabalho de pesquisa, destacando as contribuições, dificuldades e limitações, conclusão, oportunidades de trabalhos futuros e a disseminação dos resultados obtidos.

6.1 CONTRIBUIÇÕES

Esta pesquisa teve como tema a especificação de requisitos de *software* e teve como principal contribuição a proposição de um método para especificação de requisitos de *software* denominado *Problem-Based SRS*. O *Problem-Based SRS* é um método ab*rang*ente descrito por meio de um processo envolvendo atividades, especificação dos artefatos e um fluxo lógico. O método não prescreve um fluxo de trabalho específico pois pode ser conduzido de maneira sequencial ou por fluxos iterativos e incrementais.

O método proposto visa melhorar a qualidade das especificações em termos da adequação dos requisitos às intenções do cliente. Estas intenções são especificadas na forma de problemas e necessidades aos quais os requisitos devem estar associados, justificando o nome dado ao método pois a especificação baseia-se nos problemas originais do cliente e que motivam a busca por um *software*.

De acordo com os objetivos específicos definidos, esta pesquisa também trouxe as seguintes contribuições adicionais:

- **Estabelecimento de uma base conceitual para o método proposto**

A base conceitual proposta e detalhada no método *Problem-Based SRS* é composta pelos conceitos chave: domínio de problemas, domínio de necessidades, domínio de requisitos, *software glance* e visão de *software*. A interpretação destes conceitos é particular para o método e fornece ao analista de *software* um entendimento mais preciso de como especificar problemas, necessidades e requisitos e como sintetizar o *software glance* e a visão de *software*.

A base conceitual proposta auxilia o analista de *software* na condução das atividades de especificação de *software* e aumenta o sucesso do entendimento das reais intenções do cliente pois diferencia cada aspecto desde

o contexto de negócio passando pelos problemas e então definindo aspectos de *software* pelas necessidades e, por fim, especificando requisitos funcionais. A consolidação deste entendimento é através da escrita da especificação e, para isto, o método propõem nomenclaturas para uniformizar a especificação.

Os conceitos propostos relacionados ao projeto (*design*) do *software* como *software glance* e visão de *software* auxiliam o analista com aspectos não tradicionalmente observados, entretanto, são de vital importância para o sucesso de uma especificação de *software*. O conceito de *software glance* introduzido nesta pesquisa é uma das colaborações mais importantes devido a sua importância e uso fundamental para a tomada de decisão de especificação de *software*. O reconhecimento que analistas de *software* constroem soluções durante a especificação de *software* é clara utilizando o conceito de *software glance*. A "Visão de S*oftware"* é um conceito conhecido que no método é especificado com base no *software glance* e esta fundamentação auxilia o analista no detalhamento de outros aspectos da especificação do *software*.

- **Definir uma nomenclatura para a especificação**

O método *Problem-Based SRS* definiu um conjunto de conceitos que inclui as nomenclaturas para a especificação de problemas, necessidade e requisitos e a utilização de classes para os verbos de problemas e necessidades. Para a especificação de problemas foram propostas três classes de problemas possíveis: obrigação, expectativa e esperança. Para a especificação necessidades foram propostas quatro classes de necessidades possíveis: informação, controle, construção e entretenimento.

- **Propor uma técnica de mapeamento e análise entre domínios**

O método *Problem-Based SRS* propôs a aplicação de mapeamento da especificação de *software* entre três domínios específicos o de problema, necessidade e requisitos. O método auxilia o analista de *software* a mapear as intenções, necessidades e requisitos de forma que o conjunto de informação produzida seja independente funcionalmente, ou seja, que cada aspecto da solução proposta não cause interferência em outros aspectos, assim, entregando uma especificação de qualidade superior comparada com especificações tradicionais.

Como técnicas de análise o *Problem-Based SRS* ofereceu a análise de rastreabilidade para compreensão de aspectos de cobertura e conformidade e a análise utilizando fundamentos do a*xiomatic design*. A análise de rastreabilidade auxilia o analista a evitar erros comuns de especificação como a não cobertura ou conformidade dos problemas ou necessidades dos clientes. As matrizes de rastreabilidade auxiliam o analista no processo de análise e destas falhas e apresentam o grau de conformidade da especificação.

A análise com fundamentos do AD oferece ao analista um entendimento e resposta sobre os possíveis casos de falhas nas especificações que levam a especificações acopladas, semi-acopladas e redundantes. Com este entendimento o analista pode tomar melhores decisões ao longo da especificação de *software* aumentando assim as chances de sucesso posteriores a especificação de *software*. A utilização de técnicas de análise de qualidade aumenta o sucesso e guiam o analista na avaliação de especificações de *software*.

6.2 CONCLUSÃO

Este trabalho de pesquisa propôs um novo método para especificação de requisitos de *software*. Este método visa colaborar para a melhoria da qualidade da especificação de requisitos de *software*. Os benefícios observados com a utilização do método *Problem-Based SRS* são:

(*i*) Fornece uma forma sistemática para elaborar e descrever a especificação de requisitos de um *software* baseada nos problemas do cliente.

(*ii*) Esclarece e fornece conceitos distintos e notações para a definição de problemas e necessidades dos clientes ampliando a abrangência da especificação de requisitos.

(*iii*) Contribui para a melhoria da qualidade da especificação por meio do mapeamento dos conceitos entre os domínios desde os problemas até os requisitos, melhorando a precisão da cobertura dos problemas e necessidades e da justificativa das necessidades e requisitos.

(*iv*) Permite ao analista ter conhecimento do grau de dependência e do grau de satisfação da especificação produzida e indicativos de melhorias que podem ser feitas.

(*iv*) Esclarece e fornece meios de posicionar a concepção de alto nível da solução de *software* ao longo do processo de especificação na forma do *software glance* e visão de *software*.

O método *Problem-Based SRS* enriquece o processo de especificação de requisitos de *software* tornando claro como os requisitos de *software* são especificados desde a identificação dos problemas e necessidades do cliente. De fato, este trabalho pode ter um impacto significativo sobre a qualidade da especificação de requisitos de *software* permitindo oferecer o *software* correto para os *stakeholders* de um contexto específico. Adicionalmente, os conceitos e os processos propostos pelo método *Problem-Based SRS* são flexíveis o bastante para serem alinhados com as propostas discutidas no referencial teórico como GORE e VPD.

Como limitações deste trabalho, o autor considera que:

- Apesar deste trabalho apresentar o processo completo de especificação de requisitos de *software*, não foram apresentados detalhes das técnicas para conduzir cada um dos subprocessos do *Problem-Based SRS*. Por exemplo, não foram discutidas nem propostas técnicas de elicitação de requisitos que podem ser utilizadas como *workshops* e entrevistas.
- Tanto o estudo de aplicação do método quanto o experimento realizado, se limitaram a uma única solução de especificação, não permitindo usar as técnicas de análise para ilustrar um comparativo entre as qualidades de especificação de duas soluções.
- A técnica de análise de independência e conteúdo de informação (relacionadas à aplicação dos Axiomas 1 e 2) trouxeram uma adaptação e avanço sobre a teoria do AD, mas ainda são limitadas. O autor considera que estas técnicas podem ser aprofundadas e que há espaço para se descobrir outras métricas complementares à análise da satisfação das necessidades e requisitos.

6.3 TRABALHOS FUTUROS

Com base nas limitações e lacunas desta dissertação, foram identificados como trabalhos futuros:

- Aprofundar os estudos sobre o Contexto de Negócios visando detalhar e delinear o contexto de *software* dentro de um sistema;
- Realizar estudos e experimentos utilizando diferentes técnicas para cada um dos subprocessos do *Problem-Based SRS*;
- Verificar a aplicabilidade do método proposto em Engenharia de Sistemas;
- Aprofundar estudos sobre a aplicação do Axioma 2 e elaborar experimentos com grupos diferentes para comparar a qualidade das especificações através de métricas do conteúdo da informação;
- Elaborar experimentos integrando o *Problem-Based SRS* com outros métodos como GORE, VPD e Métodos ágeis.

REFERÊNCIAS

BLANK, S.; DORF, B. **The Startup Owner's Manual.** 1. ed. Pescadero, CA: K&S; Ranch., v. 1, 2012.

BOURQUE, P.; FAIRLEY, R. E. **Guide to the Software Engineering Body of Knowledge, Version 3.0**. IEEE. Piscataway, NJ, p. 346. 2014.

COHN, M. **User stories applied:** For agile software development. 1. ed. Boston: Addison-Wesley Professional, v. 1, 2009.

DARDENNE, A.; VAN LAMSWEERDE, A.; FICKAS, S. Goal-directed requirements acquisition. **Science of computer programming 20.1**, 1993.

DAVEY, B.; PARKER, K. Requirements elicitation problems: A literature analysis. **Issues in Informing Science and Information Technology**, p. 71-82, 2015. Disponivel em: <http://iisit.org/Vol12/IISITv12p071-082Davey1929.pdf>.

DAVIS, A. et al. **Effectiveness of requirements elicitation techniques:** Empirical results derived from a systematic review. 14th IEEE International Requirements Engineering Conference (RE'06). Minneapolis: IEEE. 2006. p. 179 - 188.

DO, S.-H. **Software product lifecycle management using axiomatic approach.** In Anais do ICAD2004 - 3rd International Conference on Axiomatic Design. Seoul: ICAD. 2004.

DORI, D. **Object-process methodology:** A holistic systems paradigm. Berlin: Springer Science & Business Media, 2011.

GIL, A. C. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.

GIORGINI, P. et al. Reasoning with goal models. **In Conceptual Modeling-ER**, p. 167-181, 2002.

GSMA. **Agreement for International SMS Hubbing Services 7.0**. GSMA. London, p. 73. 2012. (AA.71).

GUMMUS, B. **Axiomatic product development lifecycle.** [S.l.]. 2005.

HARUTUNIAN, V.; TATE, D.; SUH, N. **Decision making and software tools for product development based on axiomatic design theory.** CIRP Annals-Manufacturing Technology 45.1. [S.l.]: CIRP. 1996. p. 135-139.

HEITMEYER, C. **Re theory meets software practice:** Lessons from the software development trenches. IEEE International Requirements Engineering Conference (RE 2007). [S.l.]: IEEE. 2007.

HOFMANN, H. F.; LEHNER, F. Requirements engineering as a success factor in software projects. **IEEE Software**, p. 58-66, 2001.

IBM, K. C. Rational Team Concert 4.0.5, 17 Nov 2015. Disponivel em: <http://www-01.ibm.com/support/knowledgecenter/SSCP65_4.0.5/com.ibm.rational.rrm.help.doc/topics/r_vision_doc.html?lang=en-us>.

IEEE, C. S. **IEEE Std 830-1998. IEEE recommended practice for software requirements specifications.** [S.l.]. 1998.

IIBA, A. **Guide to the business analysis body of knowledge (BABOK Guide)**. International Institute of Business Analysis. Toronto, Ontario, p. 272. 2009.

INCOSE. **INCOSE Systems Engineering Handbook**. 2015. ed. Hoboken, NJ: Wiley, v. 1, 2015.

ISO. **ISO/PAS 19450:2015 Automation systems and integration - Object-Process Methodology**. ISO. [S.l.]. 2015. (19450).

ISO; IEC; IEEE. **Systems and Software Engineering - Vocabulary ISO/IEC/IEEE 24765: 2010**. The Institute of Electrical and Electronics Engineers. Piscataway, NJ. 2010.

ISO; IEC; IEEE. **Systems and Software Engineering -Requirements Engineering. Technical report. 2011**. ISO; IEC; IEEE. New York, p. 94. 2011. (29148).

JACKSON, M. A. Problem analysis using small problem frames. **South African Computer Journal**, p. 47-60, 1999.

JACKSON, M. A. Problem frames and software engineering. **Information and Software Technology**, 2005. 903–912.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **Unified Software Development Process**. Reading, Massachusetts: Addison-Wesley, v. 1, 1999.

LEFFINGWELL, D.; WIDRIG, D. **Managing software requirements:** a use case approach. 2. ed. [S.l.]: Addison-Wesley Professional, 2003.

MATTAR, F. N. **Pesquisa de Marketing - Metodologia, Planejamento, Execução e Análise**. 7. ed. Rio de Janeiro: Elsevier Campus, v. 1, 2014.

MCKEAN, E. **The new oxford American dictionary. Vol. 2.** New York: Oxford University Press, 2005.

NEILL, C. J.; LAPLANTE, P. A. Requirements engineering: the state of the practice. **IEEE Software**, p. 40-45, 2003.

NOVAKOVIC, D.; HUEMER, C. A survey on business context. **In Intelligent Computing, Networking, and Informatics**, p. 199-211, 2014.

OSTERWALDER, A. et al. **Value Proposition Design:** How to Create Products and Services Customers Want. Hoboken: John Wiley & Sons., 2014.

OSTERWALDER, A.; PIGNEUR, Y. **Modeling value propositions in e-Business**. Proceedings of the 5th international conference on Electronic commerce. New York: ACM. 2003. p. 429-436.

PEREIRA, A. M. Abordagem de especificação de requisitos baseada em projeto axiomático. **Dissertação de Mestrado, Universidade Tecnológica Federal do Paraná**, Curitiba, p. 168, 2011.

PIMENTEL, A. R. Uma abordagem para projeto de software orientado a objetos baseado na teoria de projeto axiomático. **Tese de Doutorado, Universidade Tecnológica Federal do Paraná**, Curitiba, p. 189, 2007.

PIMENTEL, A. R.; STADZISZ, P. C. Application of the independence axiom on the design of object-oriented software using the axiomatic design theory. **J. Integr. Des. Process Sci.**, p. 57–69, 2006.

RAMESH, B.; JARKE, M. Toward reference models for requirements traceability. **Software Engineering, IEEE Transactions on, v. 27, n. 1**, 2001. 58-93.

SOMMERVILLE, I. **Software Engineering 9th**. 9. ed. Boston: Pearson, 2010.

STADZISZ, P. C. et al. **Relatório de Pesquisa – Sistema MicroER**. LIT/UTFPR. Curitiba. 2016.

SUH, N. P. **The Principles of Design.** New York, USA: Oxford University Press, Inc., 1990.

SUH, N. P. Designing-in of quality through axiomatic design. **IEEE Transactions on Reliability 44.2**, Jun 1995. 256-264.

SUH, N. P. **Axiomatic design:** advances and applications. New York, USA: Oxford University Press, Inc., 2001.

SUH, N. P. **Complexity:** theory and applications. New York, USA.: Oxford University Press, Inc., 2005.

SUH, N. P.; DO, S.-H. Axiomatic design of software systems. **CIRP Annals-Manufacturing Technology**, p. 95–100, 2000.

TATE, D. A Roadmap for Decomposition: Activi2 ties, Theories, and Toolsfor SystemDesign. **Tese de Doutorado, Massachusetts Institute of Technology**, Cambridge, p. 210, Jul 1999.

THOMPSON, M. K. Improving the requirements process in axiomatic design theory. **CIRP Annals - Manufacturing Technology**, p. 115 – 118, 2013.

TRISHA., G.; PEACOCK., R. Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources. **BMJ**, 2005. 1064-1065. Disponivel em: <http://dx.doi.org/10.1136/bmj.38636.593461.68>.

VAN LAMSWEERDE, A. **Goal-oriented requirements engineering:** a roundtrip from research to practice [engineering read engineering]. Requirements Engineering Conference. Kyoto, Japan: IEEE International. 2004. p. 4-7.

WIEGERS, K.; BEATTY, J. **Software Requirements 3**. Redmond: Pearson Education. Microsoft Press, 2013.

YU, E. et al. **Social modeling for requirements engineering**. Cambridge: MIT Press, 2011.

ZOWGHI, D.; COULIN, C. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools, Engineering and Managing Software Requirements. **Engineering and Managing Software Requirements**, p. 19-46, 2005.

APÊNDICE A - COROLÁRIOS E TEOREMAS DO AD

Este anexo apresenta os corolários e teoremas extraídos de Suh (2001, 2005).

A.1 Corolários Gerais do AD

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

Corolário 2 (Minimização dos requisitos funcionais). *Minimize o número de FRs e Cs.*

Corolário 3 (Integração das partes físicas). *Integre características de projeto em uma única parte física se os FRs 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 FRs e Cs.*

Corolário 5 (Uso de simetria). *Use formas e componentes simétricos se eles estiverem consistentes com os FRs e Cs.*

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

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

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 do AD

Teorema 1 (Acoplamento devido a um número insuficiente de *DPs*). *Quando o número de DPs é menor que o número de FRs, ou ele resulta em um projeto acoplado ou FRs não poderão ser satisfeitos.*

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

Teorema 3 (Projeto redundante). *Quando existem mais DPs que FRs, o projeto ou é um projeto redundante ou é um projeto acoplado.*

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

Teorema 5 (Necessidade de um novo projeto). *Quando um conjunto de FRs é alterado pela adição de um novo FR, seja pela substituição de um dos FRs, por um novo ou pela seleção de um conjunto completamente diferente de FRs. A solução do projeto dada pelos DPs originais não pode satisfazer o novo conjunto de FRs. 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 DPs são mudados para satisfazer um dado conjunto de FRs.*

Teorema 7 (Dependência de caminho de projeto acoplados e semi-acoplados). *O conteúdo de informação de um projeto acoplado ou semi-acoplado depende da sequência pela qual os DPs são mudados e dos caminhos específicos de mudança destes DPs.*

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

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 FRs para o produto com o vetor dos DPs do produto), multiplicada pela matriz de projeto do processo de manufatura (B) (que relaciona o vetor de FRs para o processo de manufatura com o vetor dos DPs 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 FRs 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 FRs.*

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 FRs e dos DPs enquanto forem preservadas as associações entre cada FR e seus correspondentes DPs.*

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 DP for probabilisticamente independente dos outros DPs, o conteúdo de informação total do sistema é a soma da informação de todos os eventos individuais associados com o conjunto de FRs que devem ser satisfeitos.*

Teorema 14 (Conteúdo de informação de projetos acoplados versus desacoplados). *Quando o estado dos FRs é mudado de um estado para o outro*

no domínio funcional, a informação requerida para a mudança é maior para projetos acoplados do que para projetos desacoplados.

Teorema 15 (Interface projeto-manufatura). *Quando um sistema de manufatura compromete a independência dos FRs 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 FRs 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 para um projeto semi-acoplado, se a informação faltante está relacionada com elementos fora da diagonal.*

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

Teorema 19 (Robustez do projeto). *Um projeto desacoplado e um projeto semi-acoplado 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.*

A.3 Teoremas Relacionados a *Design* 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 FRs puderem ser monitoradas para permitir um controle dos FRs.*

Teorema Soft 2 (Tomada de decisões corretas na ausência do conhecimento completo para um projeto semi-acoplado com controle). *Quando o sistema de software é um projeto semi-acoplado, os FRs podem ser satisfeitos pela mudança dos DPs se a matriz de projeto é conhecida para a extensão que*

o conhecimento a respeito da sequência apropriada é fornecido, mesmo que não haja um conhecimento preciso a respeito dos elementos do projeto.

A.4 Teoremas Complexidade

Teorema C 1 (Complexidade para um sistema desacoplado com muitas partes interconectadas) A complexidade para um sistema desacoplado com muitas partes interconectadas não é, necessariamente, maior que um sistema com menos partes interconectadas, a não ser que, as interfaces entre as partes interconectadas de um sistema desacoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range.*

Teorema C 2 (Complexidade para um sistema semi-acoplado com muitas partes interconectadas) A complexidade para um sistema semi-acoplado com muitas partes interconectadas não é, necessariamente, maior que um sistema com menos partes interconectadas, a não ser que, as interfaces entre as partes interconectadas de um sistema semi-acoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range.*

Teorema C 3 (Complexidade para um sistema acoplado com muitas partes interconectadas) A complexidade para um sistema acoplado com muitas partes interconectadas é, necessariamente, maior que um sistema com menos partes interconectadas, desde que, qualquer variação nas interfaces entre as partes interconectadas de um sistema acoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range.*

Teorema C 4 (Complexidade para um sistema desacoplado com uma combinação complicada de partes) A Complexidade para um sistema desacoplado com uma combinação complicada de partes não é, necessariamente, maior que um sistema com uma menor combinação complicada, desde que, as interfaces entre as partes de um sistema desacoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range.*

Teorema C 5 (Complexidade para um sistema semi-acoplado com uma combinação complicada de partes) A Complexidade para um sistema semi-acoplado com uma combinação complicada de partes não é, necessariamente,

maior que um sistema com uma menor combinação complicada, desde que, as interfaces entre as partes de um sistema semi-acoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range*.

Teorema C 6 (Complexidade para um sistema acoplado com uma combinação complicada de partes) A Complexidade para um sistema acoplado com uma combinação complicada de partes não é, necessariamente, maior que um sistema com uma menor combinação complicada, desde que, as variações das interfaces entre as partes de um sistema acoplado não aumentem a incerteza através da redução da faixa comum entre *system range* e *design range.*

Teorema C 7 (Complexidade Imaginaria para um sistema desacoplado com uma combinação complicada de partes) A complexidade imaginária independente de tempo de um sistema desacoplado com uma combinação complicada de partes pode ser maior se os parâmetros de projeto (DPs) não são modificados na sequência de uma matriz de projeto.

Teorema C 8 (Complexidade para um sistema que atravessa muitas escalas) Quando um sistema deve ser integrado através de muitas escalas de medidas (ou tempos), este sistema deve ser dividido em pequenas subunidades, que serão integradas, para minimizar o conteúdo da informação e reduzir a complexidade real independente de tempo.

A.4 Teoremas Complexidade Relacionados a Sistemas Sócio Político Econômicos

Teorema SPE C 8 (Complexidade para um sistema Sócio Político Econômico) A complexidade para um sistema Sócio Político Econômico aumenta com o número de entidades (*i.e.,* organizações de indivíduos) que podem afetar o resultado final.

Teorema SPE C 9 (Redução da complexidade de um sistema Sócio Político Econômico) Caso todos os constituintes de um sistema social podem concordar com um conjunto de FRs e estes FRs podem ser satisfeitos independentemente a complexidade do processo de tomada de decisão pode ser reduzido quando a decisão final for tomada por uma simples entidade logo após entender e levar em consideração as incertezas introduzidas por outros constituintes do sistema.

Teorema SPE C 10 (Redução da complexidade de um sistema Sócio Político Econômico através da reinicialização e re-projeto) Quando o sistema Sócio Político Econômico está se movendo para um estado caótico, por causa da complexidade combinatória dependente de tempo, o sistema deve ser reinicializado ou re-projetado para reduzir a complexidade.

APÊNDICE B - MÉTODO UTILIZADO NO PESQUISA BIBLIOGRÁFICA

Esta subseção descreve a estratégia, métodos e etapas executadas para a extração, classificação e análise dos trabalhos que compõem a revisão bibliográfica. Com o objetivo de identificar os trabalhos relevantes, as perguntas da Quadro 32 foram criadas e tem como tópicos de interesses: fundamentos, processos, elicitação, análise e especificação. Para o AD, os tópicos de interesses são: fundamentos, requisitos funcionais e domínio do cliente. Como ponto de início foram escolhidos os trabalhos realizados pela UTFPR por serem relacionados ao tema de estudo.

#	Questão para Revisão Bibliográfica
RQ.1	Qual é o método de pesquisa para revisão bibliográfica?
RQ.2	Quais são os estudos já realizados (pela UTFPR) em *Axiomatic Design*?
RQ.3	Quais são as referências relevantes para o binômio *Axiomatic Design* e Engenharia de Requisitos de *Software*?

Quadro 33 - Questões para Revisão Bibliográfica
Fonte: O Autor.

A estratégia da pesquisa está dividida em duas fases como apresentado na Figura 38. A primeira fase envolve uma pesquisa por referências e citações e a segunda uma pesquisa em bases de dados acadêmicas.

A primeira fase tem como ponto de início os estudos executados anteriormente pela UTFPR e orientados pelos professores Paulo C. Stadzisz e Jean M. Simão. Nesta fase houve uma pesquisa sobre as referências e citações dos trabalhos de Andrey Pimentel (2007), Ana Maria Pereira (2011) e Márcio Venâncio Batista (2013) e é similar ao método *snowballing* (TRISHA. e PEACOCK., 2005).

A segunda fase é uma pesquisa nas bases de dados acadêmicas utilizando palavras-chaves.

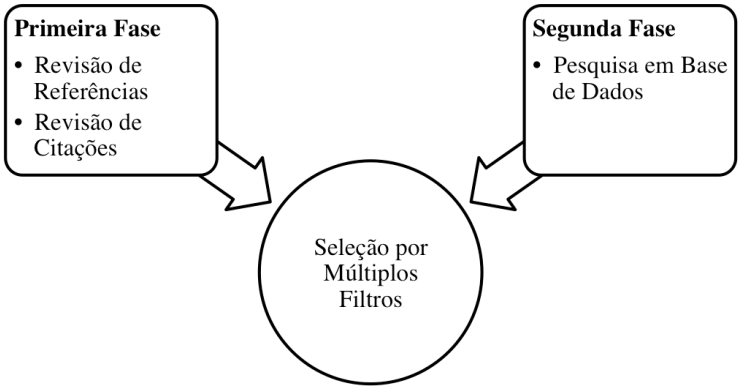


Figura 38 - Estratégia de pesquisa para revisão bibliográfica.
Fonte: O Autor.

Após a seleção inicial, todos os trabalhos escolhidos na primeira e segunda fases são submetidos aos métodos de seleção e filtragem em múltiplas etapas. Os critérios estabelecidos para os filtros são apresentados na Quadro 33. A escolha do filtro por data de publicação foi definida com base nos trabalhos iniciais como o de Suh de 1990. Procurou-se trabalhos com ênfase em: Engenharia de Requisitos e *Axiomatic Design*, Requisitos e Axiomático, Projeto de *Software* e *Axiomatic Design*.

#	Estágio	Critério de Seleção
1	Seleção geral	• Data de publicação entre Janeiro 1990 e Outubro 2014. • Publicado em Inglês ou Português. • Publicado em Conferências / Periódicos ou Livros. • Não duplicados. • Somente textos completos. • Ênfase em: Engenharia de Requisitos e *Axiomatic Design*, *Software* e *Axiomatic Design.* • Caso possível: somente periódicos revisados por pares.
2	Seleção por título e resumo	• Mesmos critérios do Estágio 1. • Tópicos de interesse em Engenharia de Requisitos: fundamentos, processos, elicitação, análise e especificação. • Tópicos de interesse em AD: fundamentos, requisitos funcionais, domínio do cliente. • Tem um fundamento empírico. • Principal foco em fases iniciais de Requisitos e/ou *Axiomatic Design.* • Estudo relacionado ao domínio de Engenharia de *Software.*

3	Análise do texto completo	• Contribuição para a discussão sobre estágios iniciais de Engenharia de Requisitos e o *Axiomatic Design.*

Quadro 34 - Critérios de inclusão
Fonte: O Autor.

A Figura 39 apresenta os detalhes da primeira fase de pesquisa. Para cada um dos trabalhos foram analisadas as referências relevantes por meio de seleção manual com leitura de título e resumo. E de modo recursivo, apresentou-se um novo nível, resultando em um conjunto de trabalhos selecionados. Por meio do portal de periódicos do capes e o ACM *digital library* foram reunidas as citações relevantes sobre os trabalhos relacionados e, também de modo recursivo, mais um nível de trabalhos relevantes foram selecionados manualmente fazendo-se a de leitura de título e resumo.

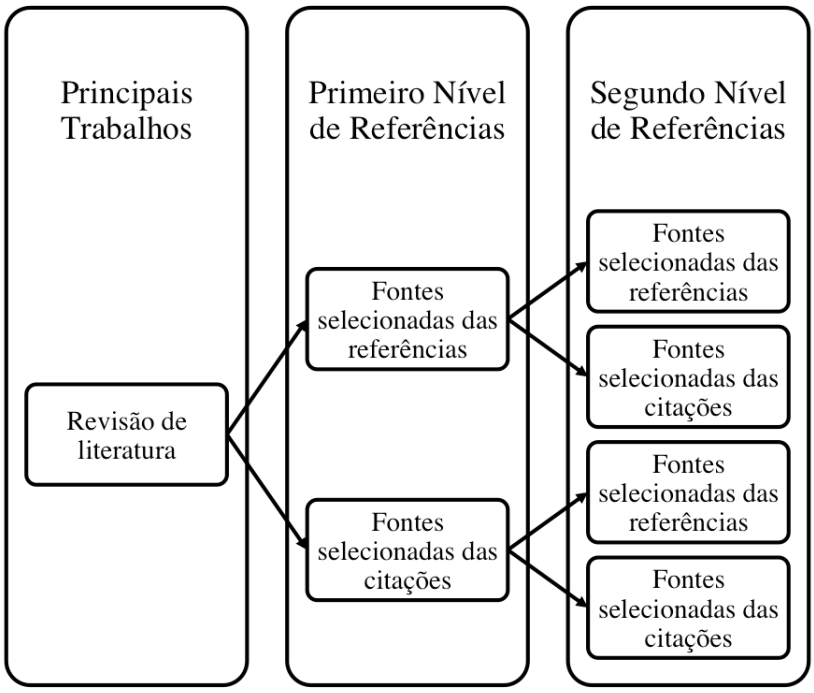


Figura 39 - Método de pesquisa da primeira fase.
Fonte: O Autor.

A segunda fase (Tabela 7) utilizou, como método, a pesquisa em bases de dados acadêmicos e, as palavras-chave. Aqui, fez-se uso das seguintes bases de dados acadêmicas e repositórios de publicações de jornais, seminários e conferências:

- Portal de Periódicos do Capes.
- ACM Digital Library.
- Publicações das Conferencias Internacionais de *Axiomatic Design* (ICAD).

- Publicações do *Workshop on Requirements Engineering* (WER).
- IEEE *International Requirements Engineering Conference* (RE).
- IEEE *Transactions* em Engenharia de *Software*.

A segunda fase de pesquisa necessitou utilizar expressões diferentes para cada base de dados. A Tabela 7 apresenta as palavras-chave utilizadas e o resultado da primeira seleção. Por ser uma base de dados de trabalhos em português e inglês, o portal de periódicos do CAPES teve a necessidade de ser pesquisado duas vezes, sendo a primeira em inglês e a segunda em português.

Tabela 7 - Palavras chaves e resultados da seleção geral da segunda fase

Base de dados	Expressões utilizadas	Total encontrado
Portal de Periódicos CAPES/MEC (Inglês)	*axiomatic+design AND requirements*	112
Portal de Periódicos CAPES/MEC (Português)	projeto+axiomático	14
ACM Digital Library	"*axiomatic design*" AND "*requirements*"	220
ICAD	Requirements *software customer domain*	64
IEEE	"*axiomatic design*"	13

Fonte: O Autor.

Foram constatados vários trabalhos duplicados como resultado da procura em múltiplas bases semelhantes, assim os mesmos foram removidos da contabilização. O mesmo caso acontece para trabalhos que continham resumo em língua inglesa e portuguesa, os mesmos foram contabilizados unicamente. Após a seleção dos trabalhos da primeira e segunda fase, o processo de seleção por múltiplos filtros foi aplicado como apresentado na Figura 40.

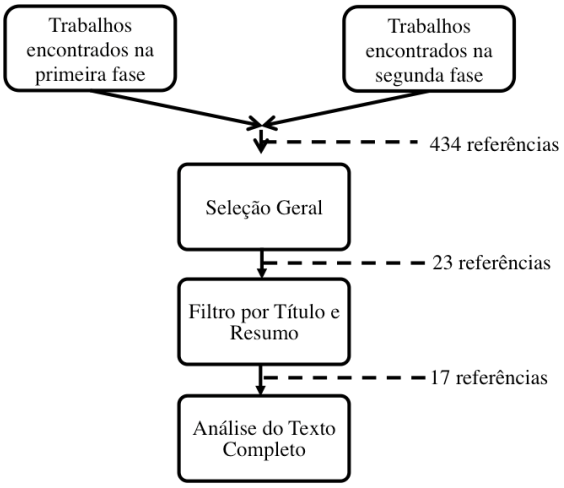


Figura 40 - Método de seleção de múltiplas etapas.
Fonte: O Autor.

A revisão sistêmica foi aplicada para as duas fases da estratégia de pesquisa que ocorreu no período entre fevereiro e outubro de 2014 e resultou nos seguintes trabalhos por fase e etapa seguindo os critérios da Tabela 7. Adicionalmente aos trabalhos encontrados nas duas fases, foram incluídos os padrões da IEEE relativos à engenharia de requisitos.

Seleção geral: o processo de pesquisa em duas fases resultou em 434 trabalhos, sendo 8 da primeira fase, 423 da segunda fase e 3 padrões da IEEE. Após a aplicação do critério de seleção geral, foram selecionados 23, sendo 14 trabalhos relacionados a *axiomatic design* e 9 relacionados a engenharia de requisitos. O Quadro 34 contém a lista dos trabalhos selecionados.

Seleção por título e resumo: a seleção por título e resumo contou com 23 trabalhos que passaram por um processo de leitura dos títulos e resumos (inglês e português) pelo autor dos quais somente 16 trabalhos foram manualmente selecionados por terem relação com fases iniciais da engenharia de requisitos ou *axiomatic design* e requisitos.

Análise do texto completo e utilização: a última fase foi a leitura e síntese dos trabalhos selecionados com objetivo de extrair os tópicos para revisão bibliográfica que constituirá da dissertação de mestrado futura do autor. O resultado da revisão está presente nas seguintes seções deste documento.

Ênfase do trabalho	**Seleção por título e resumo (23)**	**Análise do texto completo (16)**

Fundamentos de Engenharia de Requisitos	(IEEE, 1998), (NEILL e LAPLANTE, 2003), (LEFFINGWELL e WIDRIG, 2003), (COHN, 2009), (ISO, IEC e IEEE, 2011), (ISO, IEC e IEEE, 2010), (WIEGERS e BEATTY, 2013), (BOURQUE e FAIRLEY, 2014), (JACOBSON, BOOCH e RUMBAUGH, 1999)	(IEEE, 1998), (NEILL e LAPLANTE, 2003), (LEFFINGWELL e WIDRIG, 2003), (ISO, IEC e IEEE, 2010), (ISO, IEC e IEEE, 2011), (WIEGERS e BEATTY, 2013), (BOURQUE e FAIRLEY, 2014), (JACOBSON, BOOCH e RUMBAUGH, 1999)
Fundamentos de *Axiomatic Design*	(SUH, 1990), (SUH, 1995), (SUH, 2001), (SUH, 2005), (TATE, 1999), (HARUTUNIAN, TATE e SUH, 1996),	(SUH, 1990), (HARUTUNIAN, TATE e SUH, 1996), (SUH, 2001), (SUH, 2005), (TATE, 1999),
Software* e *Axiomatic Design	(SUH e DO, 2000), (DO, 2004), (PIMENTEL e STADZISZ, 2006), (PEREIRA, 2011)	(PIMENTEL, 2007)
Requisitos e Projetos Axiomáticos	(GUMMUS, 2005) (HEITMEYER, 2007), (PEREIRA, 2011), (THOMPSON, 2013)	(PEREIRA, 2011), (THOMPSON, 2013)

Quadro 35 - Revisão de referências selecionadas
Fonte: O Autor.

A revisão bibliográfica é utilizada como base para o referencial teórico da Seção 2 e está dividido em dois tópicos: engenharia de requisitos e *axiomatic design*. O tópico de engenharia de requisitos utiliza os resultados obtidos por revisões bibliográficas e de autores que foram utilizados como referência pelos trabalhos anteriores selecionados na primeira fase da pesquisa executada. O tópico *axiomatic design* utiliza o resultado da primeira e segunda fase (Figura 40) da pesquisa de trabalhos e apresenta a discussão sobre o domínio do cliente no qual reside a engenharia de requisitos na área de engenharia de *software*.

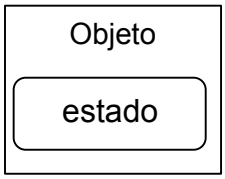
APÊNDICE C - Notação *Object Process Diagram*

Neste apêndice é apresentado uma breve descrição sobre *Object Process Methology* (OPM) que é uma abordagem holística para estudar e desenvolver sistemas. OPM integra paradigmas de estrutura da orientação a objetos e o comportamento da orientação a processos e foi desenvolvido por Dov Dori em 1995. Recentemente (final de 2015) o OPM foi aceito pela ISO como uma PAS (*Publicly Available Specification*) utilizando a referência: *ISO/PAS 19450:2015(en) Automation systems and integration - Object-Process Methodology* (ISO, 2015).

Os dois elementos da ontologia OPM são entidades (objetos *stateful* e processos) e ligações. Objetos são coisas que existem e podem ser físicos ou informacionais enquanto processos são coisas que transformam objetos. As ligações podem ser estruturais ou procedurais. Uma ligação estrutural expressa uma ligação estática entre um par de entidades. Ligações procedurais conectam entidades para descrever o comportamento de um sistema. O comportamento pode ser representando de três formas: Processos podem transformar objetos; Objetos podem habilitar processos; Objetos podem chamar eventos que invocam processos (DORI, 2011).

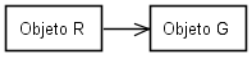
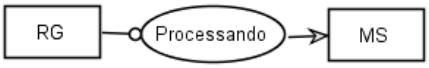
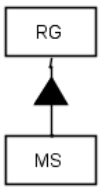
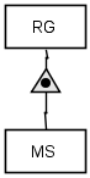
Cada sistema modelado com OPM tem como resultado produzido um conjunto de OPDs (*Object-Process Diagrams*) e uma coleção de sentenças correspondentes escritas em OPL (*Object-Process Language*). OPD e OPL são modalidades semanticamente equivalentes sendo uma gráfica e outra textual (DORI, 2011). A modalidade OPD parte do princípio da teoria cognitiva, o Quadro 36 apresenta os principais elementos do OPD.

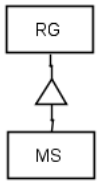
Representação	Descrição
Objeto	Objeto é uma coisa que existe ou pode existir fisicamente ou informacional.
Processo	Processo é uma coisa que transforma um ou mais objetos: (*i*) Criando, (*ii*) Destruindo e (*iii*) Afetando

	Um processo pode afetar um Objeto através da modificação de seu estado. Objetos são *statefull* portanto tem estados.

Quadro 36 - Elementos OPD
Fonte: Dori (2003)

O Quadro 36 apresenta de forma simplificada o relacionamento entre objetos, processos e estados. Os exemplos utilizados neste trabalho foram elaborados com a ferramenta de modelagem *Object-Process Case Tool* (OPCAT). Com o OPCAT é possível criar modelos de forma bidirecional (gráfico ou textual) e independente da opção escolhida é possível sincronizar as duas formas automaticamente. OPCAT ajuda no controle de visibilidade dos detalhes dos diagramas utilizando as operações de *zoom-in* e *zoom-out*. O *zoom-in* apresenta os detalhes de comportamento de um processo enquanto o *zoom-out* apresenta uma visão alto nível de abstração.

Representação	Descrição
	Relação entre objeto origem e objeto destino. Objeto R é relacionado com Objeto G.
	Processo “Processando” requere Objeto “RG”. Processo “Processando” cria Objeto MS.
	Agregação - Participação Objeto RG consiste do Objeto MS.
	Exibição - Caracterização Objeto RG exibe Objeto MS.

	Generalização - Especialização Objeto RG é um Objeto MS.

Quadro 37 - Relações Estruturais OPM
Fonte: Dori (2011)