PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

CURSO DE ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

LUCIANA DOS ANJOS CHAVES

Análise de Técnicas de Modelagem e Especificação de Requisitos

São Paulo, 2012

LUCIANA DOS ANJOS CHAVES

Análise de Técnicas de Modelagem e Especificação de Requisitos

Monografia apresentada ao Curso de Especialização em Engenharia de Software da Pontifícia Universidade Católica de São Paulo, como requisito parcial para obtenção do título de Especialista em Engenharia de Software, orientado pelo Prof. Mestre José Paulo Levandovski Papo.

São Paulo, 2012

Análise de Técnicas de Modelagem e Especificação de Requisitos

LUCIANA DOS ANJOS CHAVES

Monografia defendida e aprovada, em *28 de Dezembro de 2012*, pela banca examinadora:

Professor Mestre José Paulo Levandovski Papo

Orientador

DEDICATÓRIA

Dedico este trabalho a todos aqueles que contribuíram de forma direta ou indireta para conclusão do mesmo.

AGRADECIMENTOS

Ao meu orientador Professor Mestre José Paulo Levandovski Papo por todo apoio, incentivo e paciência ao longo do trabalho.

Aos meus amigos e colegas de trabalho, que indicaram e cederam livros durante toda a pesquisa bibliográfica.

Aos meus pais, Roseli e Sérgio, pelo incentivo constante, não só no desenvolvimento deste trabalho, mas no meu desenvolvimento pessoal, educacional e profissional durante toda minha vida.

Ao meu marido, Vinicius Sigiani, em especial, por toda a paciência e compreensão ao longo da nossa vida, e em particular, durante o desenvolvimento deste trabalho.

“O conhecimento torna a alma jovem e diminui a amargura da velhice. Colhe, pois, a sabedoria. Armazena suavidade para o amanhã.”

Leonardo da Vinci

Um ladrão rouba um tesouro, mas não furta a inteligência. Uma crise destrói uma herança, mas não uma profissão. Não importa se você não tem dinheiro, você é uma pessoa rica, pois possui o maior de todos os capitais: a sua inteligência. Invista nela. Estude!

Augusto Cury

"Com tanta tecnologia hoje em dia, é tão mais difícil ajudar o próximo na esquina."

Bárbara Campos

RESUMO

No processo de desenvolvimento de software, os requisitos são um dos elementos mais importantes, pois nenhuma outra parte do projeto é tão difícil de corrigir e prejudicial ao sistema resultante caso seja feita incorretamente. Requisitos de software definem o que um sistema deve fazer, mas sem descrever como ele deve ser feito. São as funções, características e as propriedades de um sistema. Focando no problema de documentação ou modelagem dos requisitos, as organizações, na sua maioria, têm grande dificuldade de identificar e adotar um modelo adequado para realizar a especificação de requisitos e acabam por gerar documentações incompletas ou ilegíveis, pois os modelos utilizados não se adequam ao tipo de projeto ou ao formato da organização. Um modelo deve estabelecer as características de funcionamento e comportamento de um sistema. É através do modelo que um software é criado, pois esse facilita o entendimento do projeto. Dessa forma, há uma necessidade decisiva na escolha de um processo adequado para a modelagem e a especificação dos requisitos. Este trabalho busca apresentar, analisar e comparar diversas técnicas de modelagem de requisitos com exemplos de suas respectivas documentações, prós e contras através de uma pesquisa descritiva e comparativa, na forma de estudos descritivos com base na pesquisa bibliográfica. Permitindo ao leitor conhecer uma variedade de técnicas de modelagens e os formatos mais adequados para cada tipo de projeto.

Palavras-chave: Especificação de requisitos, modelagem, requisitos, técnicas de modelagem de requisitos.

ABSTRACT

In the software development process, requirements are one of the most important elements, because no other part of the project is so difficult to correct and detrimental to the resulting system if done incorrectly. Software requirements define what a system should do, but without describing how it should be done. Are the system functions, features and properties. Focusing on the problem of modeling and documentation of requirements, organizations, mostly, have great difficulty in identifying and adopting an appropriate model to perform the requirements specification and end up generate incomplete or illegible documentation, because the models used are not appropriate to the project type or organization shape. A model must establish the operating characteristics and the system behavior. Is through a model that a software is created, because this facilitates the understanding of the project. Thus, there is a crucial need in choosing a suitable process for modeling and requirements specification. This work intends to present, analyze and compare several techniques for modeling requirements with examples of their documentation, pros and cons using a descriptive and comparative research, in the descriptive studies form based on bibliographic research. Allowing the reader to know a range of modeling techniques and the most appropriate formats for each type of project.

Keywords: Requirements specification, modeling, requirements, requirements modeling techniques.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ISO	*International Standards Organization*
PSA	*Problem Statement Analyzer*
PSL	*Problem Statement Language*
REVS	*Requirements Engineering and Validation Systems*
RSL	*Requirements Statement Language*
SADT	*Structured Analysis and Design Technique*
SREM	*Software Requirements Engineering Methodology*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

SUMÁRIO

1. Introdução

Este capítulo apresenta a motivação e a descrição do problema, os objetivos do trabalho, a metodologia adotada, os resultados esperados e a organização do trabalho.

1.1 Motivação

De acordo com Brooks em seu artigo *No Silver Bullet* (1987), a parte mais difícil na construção de um sistema de software é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil quanto estabelecer requisitos técnicos detalhados, incluindo todas as interfaces para pessoas, máquinas e outros sistemas de software. Nenhuma outra parte do trabalho prejudica o sistema resultante caso seja feita incorretamente. Nenhuma outra parte é tão difícil de corrigir mais tarde.

Pode-se considerar também, que requisitos, são os fatores mais comuns do insucesso de projetos de software. Segundo Leffingwell e Widrig (1999), dada a frequência dos erros de requisitos e o efeito multiplicativo do fator "custo para consertar", é fácil prever que os erros de requisitos contribuirão para a maioria - muitas vezes de 70 por cento ou mais - dos custos de retrabalho. Já que retrabalho geralmente consome 30 por cento a 50 por cento de um projeto típico (LEFFINGWELL; WIDRIG, 1999, citado em BOEHM, 1988). Conclui-se então que os erros de requisitos podem consumir de 25 por cento a 40 por cento do orçamento total do projeto.

Além disso, de acordo com a pesquisa realizada pelo *Standish Group, Chaos Report* (1995), requisitos incompletos está no topo da lista como um dos motivos porque os projetos são comprometidos e, finalmente, cancelados, conforme reproduzido na tabela 1.

Tabela 1 – Opiniões sobre porque os projetos são prejudicados e cancelados

Project Impaired Factors	**% of Responses**
1.Incomplete Requirements	13,10%
2.Lack of User Involvement	12,40%
3.Lack of Resources	10,60%

4.Unrealistic Expectations	9,90%
5.Lack of Executive Support	9,30%
6.Changing Requirements & Specifications	8,70%
7.Lack of Planning	8,10%
8.Didn't Need It Any Longer	7,50%
9.Lack of IT Management	6,20%
10.Technology Illiteracy	4,30%
Other	9,90%

Fonte: Standish Group, Chaos Report (1995)

Dessa forma, pode-se considerar a extração e documentação dos requisitos uma das fases mais importantes do processo de desenvolvimento de software.

Focando no problema de documentação, existem para tanto as técnicas de modelagem de requisitos funcionais. De acordo com o UML Guia do Usuário (BOOCH; RUMBAUGH; JACOBSON, 2006) com a modelagem, alcançamos quatro objetivos:

1. Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja.
2. Os modelos permitem especificar a estrutura ou o comportamento de um sistema.
3. Os modelos proporcionam um guia para a construção do sistema.
4. Os modelos documentam as decisões tomadas.

Uma vez compreendidos, analisados e aceitos, os requisitos devem ser documentados com um nível de detalhamento adequado, produzindo a especificação de requisitos de software, para tanto pode ser utilizada a linguagem natural e/ou diagramas (BLASCHEK, 2002).

A técnica mais popularmente conhecida são os Casos de Usos, que nos últimos anos, tornaram-se bem estabelecido como uma das técnicas fundamentais de análise orientada a objetos. No entanto, devido à toda essa popularidade, tem havido pouca discussão sobre as limitações e os perigos potenciais associados com casos de uso, levando a uma aceitação acrítica desta técnica (FIRESMITH, 1996), diminuindo a utilização e conhecimento de outras técnicas que podem ser alternativas mais adequadas de acordo com o tipo de projeto, empresa e equipe.

1.2 Objetivos

Este trabalho tem como objetivo apresentar, analisar e comparar diversas técnicas de modelagem de requisitos com exemplos de suas respectivas documentações, prós e contras. Permitindo ao leitor conhecer uma variedade de técnicas de modelagens e os formatos mais adequados para cada tipo de projeto.

A modelagem é uma parte central de todas as atividades que levam à implantação de um bom software (BOOCH; RUMBAUGH; JACOBSON, 2006).

1.3 Resultados Esperados

Este trabalho resultará em uma análise comparativa de técnicas de modelagem e especificação de requisitos, com a finalidade de apoiar ao leitor na escolha da técnica mais adequada para cada tipo de projeto.

Nas metodologias clássicas, o processo de requisitos é fundamental para o sucesso de um projeto de software, um bom documento de requisitos possibilita estimativas de custos razoavelmente precisas, um cronograma de execução que não deverá sofrer variações significativas e usuários podem participar ativamente do processo de validação do software (SAYÃO; STAA; LEITE, 2003).

1.4 Método de Trabalho

A finalidade deste trabalho, como definido no item 1.2, é levantar e apresentar as principais técnicas de modelagem de requisitos. Não serão discutidas a gerência e técnicas de extração de requisitos, que justificariam um trabalho à parte.

Para a realização deste trabalho será utilizado o tipo de pesquisa descritiva e comparativa, na forma de estudos descritivos, que permitem a exposição das características, propriedades e relações existentes na comunidade, grupo ou realidade pesquisada. Os estudos descritivos, assim como os exploratórios, favorecem as tarefas de formulação clara do problema a e da hipótese como tentativa da solução (RAMPAZZO, 2005).

Como estratégia, juntamente coma a pesquisa descritiva, será utilizada a pesquisa bibliográfica, pois qualquer espécie de pesquisa, em qualquer área, supõe e exige uma pesquisa bibliográfica prévia. A pesquisa bibliográfica procura explicar

um problema a partir de referências teóricas publicadas, como por exemplo, em livros, revistas, etc (RAMPAZZO, 2005).

Abaixo será apresentada a visão geral da metodologia e estratégias serão utilizadas para o desenvolvimento deste trabalho:

1. Pesquisa e revisão bibliográfica: realizar a consulta de bases bibliográficas e documentais sobre o tema. Fazer uma revisão bibliográfica prévia para investigar se existem recursos suficientes para desenvolvimento do trabalho no prazo proposto, identificando:
    a. Base técnica para orientar o leitor a respeito de requisitos;
    b. Principais técnicas de modelagem de requisitos;
    c. Prós e contras das técnicas apresentadas;
2. Pesquisa descritiva e comparativa: analisar as informações obtidas, desenvolvendo o tema, procurando propor diversas explicações para as causas dos problemas. Demonstrar as relações entre as explicações que procuram contribuir para solucionar o problema, apresentando:
    a. Análise comparativa entre as técnicas propostas, demonstrando os motivos que levam algumas técnicas a serem mais utilizadas do que as outras;
    b. Exemplos de documentações e diagramas das técnicas de modelagem apresentadas.

1.5 Organização do trabalho

O trabalho está organizado em capítulos, divididos da seguinte maneira:

No capítulo 1, Introdução, é apresentada a visão geral deste trabalho. Nele são discutidos os problemas/motivação, objetivos, resultados esperados, metodologia e a organização do trabalho.

No capítulo 2, Fundamentos e Estudos da Arte, é apresentada a base técnica para que o leitor possa entender este trabalho, detalhando os principais conceitos sobre requisitos de software.

No capítulo 3, Técnicas de Modelagem de Requisitos, são apresentadas as principais técnicas de modelagem de requisitos e uma analise comparativa entre as mesmas.

No capítulo 4, Considerações finais, é apresentada a conclusão e sugestões para trabalhos futuros.

2. Fundamentos e Estudo da Arte

Este capítulo apresenta a base técnica para que o leitor possa entender este trabalho. O item 2.1 apresenta os assuntos que serão abordados no capítulo. O item 2.2 apresenta o conceito de requisitos de software. O item 2.3 apresenta o conceito de modelagem de requisitos. O item 2.4 apresenta o conceito de documentação, análise e especificação de requisitos. O item 2.5 apresenta as considerações finais deste capítulo.

2.1 Introdução

Os conceitos que serão apresentados são: requisitos de software, modelagem de requisitos, documentação, análise e especificação de requisitos.

2.2 Requisitos de Software

Requisitos de software definem o que um sistema deve fazer, mas sem descrever como ele deve ser feito. São as funções, características e as propriedades de um sistema.

Segundo Larman (2005), requisitos são capacidades e condições as quais o sistema - e em termos mais amplos, o projeto - deve atender.

Um requisito consiste em uma declaração sobre um produto pretendido que específica o que ele deveria fazer ou como deveria operar. Um dos objetivos da atividade de estabelecimento de requisitos é torná-los o mais específicos, não-ambíguos e claros possível (PREECE; ROGERS; SHARP, 2002).

Requisitos de software ainda são definidos como características funcionais e não funcionais que o sistema precisa apresentar (MARTINS, 2007). Desta forma, requisitos podem ser classificados em dois tipos principais: funcionais e não funcionais.

Um requisito não funcional descreve um aspecto (não funcional) que o sistema deve satisfazer (RAMOS, 2006). Esse tipo de requisito definem aspectos como desempenho, usabilidade, confiabilidade, segurança, entre outros. Além disso, podem definir restrições e qualidades sobre os requisitos funcionais.

Um requisito funcional descreve uma determinada ação (ou função) que o sistema deve suportar (RAMOS, 2006). Os requisitos funcionais são a base para todo o projeto. Através deles é possível definir o escopo, tamanho e o prazo do projeto.

De acordo com Rezende (2005), é com base na descrição das necessidades do cliente ou dos requisitos funcionais do sistema, que a equipe do projeto pode especificar efetivamente o sistema de informação, suas funções, desempenho, interfaces e restrições, conforme as fases e subfases da metodologia de desenvolvimento de sistemas utilizada.

Requisitos descrevem os processos de negócio e sua interação com o produto de forma adequada. Os requisitos funcionais são todos aqueles relacionados com o comportamento de um determinado produto do projeto, ou seja, é tudo aquilo que descreve o que o produto deve fazer (MELO, 2012).

2.3 Modelagem de requisitos

Um modelo é uma abstração da realidade. É uma tentativa de estabelecer as características importantes e distintivas de alguma coisa para que possamos apreender imediatamente sua especificidade (OUCHI, 1986).

Um modelo para um sistema deve representar de forma simplificada as interações, funções e partes de um sistema. Esse modelo deve ser mais simples do que o sistema real (MEDINA; CHWIF, 2010). Se o modelo apresenta uma complexidade maior do que a do próprio do sistema, não temos um modelo, mas sim, um problema. Isso porque a intenção principal da modelagem é capturar o que realmente é importante no sistema para a finalidade em questão.

A modelagem de requisitos é a representação em forma gráfica, visual, matemática e/ou textual dos requisitos do sistema que se deseja implementar.

De acordo com Castro (2001), a modelagem de requisitos deve:

- Conter uma descrição que abrange todas as funções.
- Representar algumas pessoas, coisas físicas e conceitos importantes para o entendimento do analista sobre o que acontece no domínio da aplicação.
- Mostrar conexões e interações entre estas pessoas, coisas e conceitos.
- Mostrar a situação do negócio em detalhes para avaliar os possíveis projetos.
- Ser organizado para ser utilizado mais tarde para projetar o software.

2.4 Documentação, Análise e Especificação de Requisitos

De forma objetiva, podemos dizer que o documento de requisitos define o que o sistema irá fazer (AMBLER; CHWIF, 2002). Ele pode ser mais resumido ou composto por regras de negócio, modelagem, especificações e até protótipos de interface simples. Deste modo, normalmente, é nesse documento que a modelagem e especificação de requisitos são apresentadas, sendo considerados os principais artefatos da documentação de requisitos.

A análise de requisitos é o processo de entender, e colocar no papel, uma declaração do que uma aplicação destina-se a fazer depois de construída (BRAUDE, 2004).

O Guia BABOK (2011), define analise de requisitos como a atividade que descreve as tarefas e técnicas utilizadas por um analista de negócios para analisar requisitos declarados, no intuito de definir as capacidades requeridas de uma solução potencial para atender as necessidades das partes interessadas.

A especificação é o principal documento com respeito à obtenção e manutenção da qualidade, qualquer que seja o produto ou serviço. Sem ela, não existe base para o controle de qualidade, como exemplo disso, não podemos estar seguros de que um par de sapatos de um determinado tamanho comprado no Reino Unido seja do mesmo tamanho de um par comprado nos EUA, França ou na Austrália. Sem a especificação, nunca poderíamos ter confiança de obter uma peça de reposição para um automóvel que se ajustasse de modo perfeito apenas com a citação de um número, ou que um código de barras repetisse a mesma informação cada vez que fosse analisado (OAKLAND, 1994).

A *International Standards Organization* (ISO) define especificação na ISO 8402 (1994) como um documento que deve referenciar ou incluir desenhos, padrões ou outros documentos relevantes e indicar os meios e critérios pelos quais a conformidade possa ser verificada.

O fato de se ter uma boa documentação do software, traz várias vantagens para o desenvolvedor na hora de prestar manutenção ao programa, pois através dos detalhes mostrados nesse documento é mais fácil entender o problema e modificá-lo (DEZERBELLES, 2008).

2.5 Considerações Finais

Neste capítulo foram apresentados alguns conceitos que são importantes para o entendimento e leitura deste trabalho. Os requisitos definem o que deve ser feito em um sistema, mas sem especificar como, ou seja, sem definir, por exemplo, a linguagem de programação ou plataforma à ser utilizada e são de extrema importância em um projeto de software. Uma definição simples para requisitos é que eles, basicamente, determinam as necessidades dos *stakeholders*.

Por análise entendemos a tarefa de levantar, descrever e até negociar os requisitos de um sistema, definindo de que forma o software deve funcionar para atender as expectativas de todos que nele possuem algum interesse.

Um modelo estabelece as características de funcionamento e comportamento de um sistema. É através do modelo que um software é criado, pois esse facilita o entendimento do projeto. Um modelo deve ser considerado para o software assim como uma maquete é considerada para uma construtora: através dela, todos os envolvidos (engenheiro, corretor, cliente) sabem o produto que vão construir ou vender ou comprar.

A especificação é uma representação documentada, onde cada requisito é devidamente descrito e refinado. Ela descreve em detalhes o que o programa deverá fazer depois de pronto, complementando o modelo de requisitos utilizado.

O próximo capítulo apresenta, analisa e compara diversas técnicas de modelagem de requisitos.

3. Técnicas de Modelagem de Requisitos

Este capítulo apresenta as técnicas de modelagem de requisitos que serão analisadas e comparadas neste trabalho.

O item 3.1 apresenta a técnica de modelagem de requisitos Casos de Uso. O item 3.2 apresenta a técnica de modelagem de requisitos SADT. O item 3.3 apresenta a técnica de modelagem Redes de Petri. O item 3.4 apresenta a técnica de modelagem SREM. O item 3.5 apresenta a técnica de modelagem *User Stories*. O item 3.6 apresenta uma análise comparativa entre as técnicas exibidas neste capítulo.

3.1 Casos de Uso

A ideia de casos de uso para descrever requisitos funcionais foi introduzida, em 1986, por Ivar Jacobson, um dos principais contribuintes da UML (LARMAN, 2005). A definição formal de caso de uso, segundo a UML, é um conjunto de sequências de ações que um sistema desempenha para produzir um resultado observável de valor a um ator específico (JACOBSON, 1992).

Casos de uso são utilizados para descrever os requisitos funcionais do sistema através de um conjunto de cenários relacionados, onde cada um deles capta um curso específico de interações que ocorrem entre um ou mais atores e o sistema.

Entre as técnicas que serão apresentadas, os casos de uso é a mais popular, pois é representada por uma linguagem natural que facilita o seu entendimento por todas as partes interessadas, o que abrange, inclusive, o cliente final.

A próxima seção apresenta a definição de cada um dos itens utilizados para formar um diagrama de caso de uso.

3.1.1 Diagrama de Casos de Uso

O principal diagrama usado na UML é o diagrama de caso de uso. Ele fornece um modo de descrever uma visão externa de um sistema de informações e suas interações com o mundo, representando uma visão de alto nível de funcionalidade (FURTADO, 2002).

Um diagrama de caso de uso pode ser formado por:

- Caso de uso: é um desenho em forma oval e contém dentro dele o nome do caso de uso. Esse nome deve ser algo que represente uma ação, geralmente escrito como um verbo no infinitivo, bons exemplos seriam “Criar Item” e “Devolver Item”.

Figura 1. Diagrama de Caso de Uso

- Ator: O ator é um agente que interage com o sistema e representa um papel. Um papel pode ser representado por seres humanos, um grupo, máquina e até outros sistemas. Ele é representado pelo desenho de uma pessoa, como um “boneco de palito”.

  É importante ressaltar que um ator representa um papel e não um usuário individual. Por exemplo, “João”, não seria um nome adequado para um ator e sim o papel que ele representa, como por exemplo, “Professor”.

Figura 2. Ator do Diagrama de Caso de Uso

- Interações: é a comunicação entre casos de usos e atores. São os estímulos recebidos pelo sistema. As interações podem ser divididas em diversos tipos:
  - Associação: apresenta uma comunicação entre os atores e os casos de usos, mostrando quais atores estão ligados a quais casos de usos.

    É representada através de um arco simples.

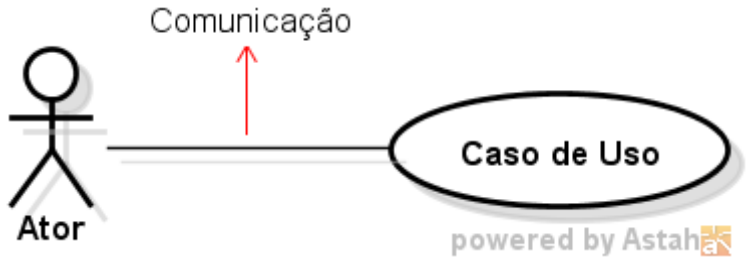


Figura 3. Interação do tipo Associação

- Inclusão: Tipo de interação usada somente entre casos de uso, ela inclui outro caso de uso na interação, ou seja, ele é composto e precisa de outro caso de uso. Quando o caso de uso principal é chamado, com a interação do tipo inclusão, sempre o caso de uso secundário será executado obrigatoriamente.

  É representada por um arco pontilhado com a descrição <<*include*>> (UML 1.4) ou <<*uses*>> (UML 1.3).

  Um exemplo de utilização da interação tipo inclusão, em um sistema de restaurante *delivery*, onde a identificação do telefone do cliente é obrigatória e a inicialização do pedido também, tudo de forma automatizada, logo quando o cliente liga, seria então o caso de uso principal “Identificar Ligação” e o caso de uso de inclusão “Iniciar Pedido”.

Figura 4. Interação do tipo Include

- Extensão: Tipo de interação usada somente entre casos de uso, ela estende outro caso de uso na interação. Nesse caso, um caso de uso utiliza opcionalmente o outro caso de uso, ao contrário do tipo de interação Inclusão.

  É representada por um arco pontilhado com a descrição <<*extend*>>.

  Utilizando o mesmo exemplo do item acima, em um sistema *delivery*, onde a identificação do telefone do cliente é obrigatória e

a inicialização do pedido não, pois o pedido só deve ser iniciado se o cliente já for cadastrado, logo quando o cliente liga, seria o caso de uso principal “Identificar Ligação” e o caso de uso de extensão “Iniciar Pedido”, pois o pedido só será iniciado automaticamente se o cliente já possuir um cadastro.

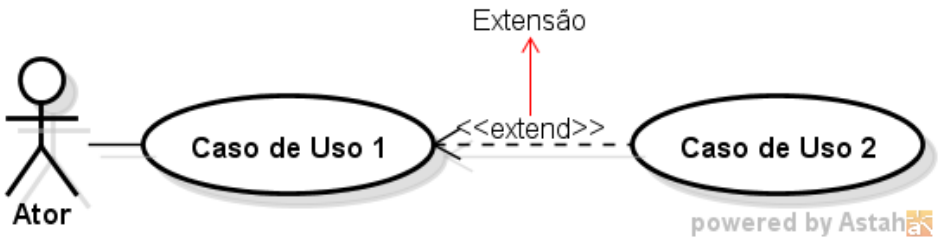


Figura 5. Interação do tipo Extensão

- Generalização: Também chamada de herança, é uma interação usada entre casos de uso e entre atores.

  A generalização entre casos de uso é utilizada para dividir um supertipo em subtipos, sendo que o caso de uso filho herda o comportamento do caso de uso pai.

  É representado por uma seta contínua que aponta do filho para o caso de uso pai.

  Um exemplo para uso de generalização entre casos de uso é um caso de uso pai “Efetuar Pagamento” com os casos de usos filhos “Pagar em Dinheiro” e “Pagar em Cartão”. Cada um dos casos de uso filhos tem sua particularidade no processo de pagamento, mas os dois possuem comportamentos similares e retornam o objetivo do pai em efetuar o pagamento.

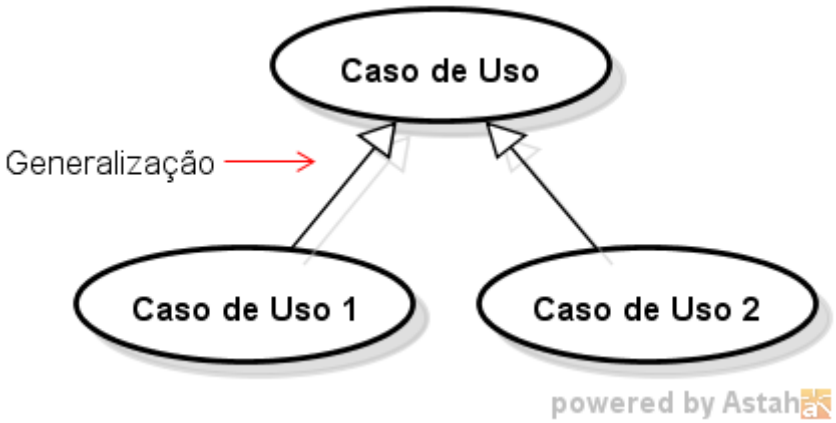


Figura 6. Interação do tipo Generalização entre Casos de Uso

A generalização entre atores é utilizada para indicar que o ator filho pode realizar todas as funcionalidades do ator pai e, se definido, ele também poderá realizar funções adicionais.

É representado por uma seta contínua que aponta do filho para o ator pai.

Um exemplo para uso de generalização entre atores é um ator pai “Funcionário” que realiza as funções (casos de usos) “Efetuar Pedido” e “Adicionar Item” o ator filho “Gerente” herda todas as funções do pai, ou seja, “Efetuar Pedido” e “Adicionar Item” e ainda possui uma função adicional “Devolver Produto” que só pode ser efetuada pelo gerente.

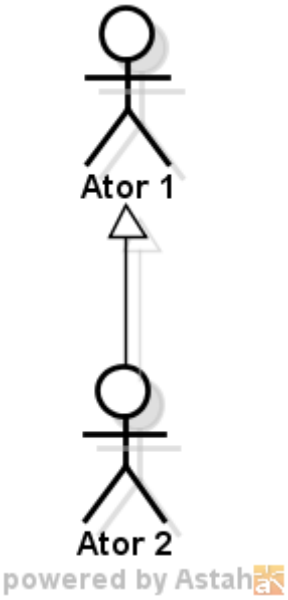


Figura 7. Interação do tipo Generalização entre Atores

3.1.2 Especificação de Casos de Uso

Especificar os casos de uso requer descrever o fluxo de eventos em detalhes, incluindo como ele começa, termina e interage com os atores. Pode ser constituído por dois passos (AMO; LOIC; PÉREZ, 2005):

1. Desenhar um diagrama de estados para cada caso de uso.
2. Fazer uma descrição textual com as pré-condições, pós-condições, caminhos básicos e alternativos do caso de uso.

Dessa forma, a especificação detalha a modelagem de caso de uso, permitindo ao leitor entender todos os caminhos possíveis (seja de sucesso ou insucesso)

daquele requisito, comportando a validação dos fluxos, mensagens, resultados e regras de negócio.

A descrição de casos de uso não possui um padrão, ou seja, uma forma única de ser escrita. Ela pode ser desenvolvida através de uma descrição informal como estórias ou divida através de pré, pós-condições e outros ou uma combinação das duas formas.

Abaixo é apresentada uma lista dos possíveis itens que podem estar contidos em uma especificação de casos de uso, de acordo com Bezerra (2002):

1. **Nome** do caso de uso.
2. **Identificador**, um código que identifique o caso de uso.
3. **Importância** do caso de uso.
4. **Sumário**, pequena descrição do caso de uso.
5. **Ator Primário**, aquele que inicia o caso de uso.
6. **Atores Secundários**, outros atores que participem do caso de uso.
7. **Pré-condições** para que o caso de uso aconteça.
8. **Fluxo Principal** são os passos que ocorrem em um cenário perfeito.
9. **Fluxos Alternativos** são as tomadas de decisões diferentes das do fluxo principal, tomadas pelos atores para alcançar o objetivo.
10. **Fluxos de Exceção** são os acontecimentos inesperados que podem vir a ocorrer na interação entre caso de uso e atores.
11. **Pós-condições**, como deve ser o estado do sistema depois de efetuada a tarefa.
12. **Regras de Negócio**
13. **Histórico**
14. **Notas de Implementação** sobre como o caso de uso pode ser implementado.

Abaixo é apresentado um exemplo simples de especificação de casos de uso (QIDIGITAL, 2011):

Especificação do Caso de Uso Controlar Figura

Este caso de uso permitirá ao professor cadastrar, consultar, excluir e alterar Figuras.

Fluxo de Eventos

Este caso de uso inicia-se depois que o usuário acessar o site do Repositório de Metáforas e selecionar o menu Figura.

Fluxo Básico

1) O sistema o direciona para uma interface gráfica referente às Figuras.

2) O sistema exibirá uma interface listando todas as Figuras já registradas permitindo consultá-las, alterá-las ou excluí-las.
3) O usuário clica no link Cadastrar nova Figura.
4) O sistema gera automaticamente o código da Figura.
5) O usuário preenche os campos e clica no botão Cadastrar.

Fluxos Alternativos

Consultar Figura

1) Após o ponto 2 do Fluxo Básico, o usuário clica no link Editar relacionado à Figura desejada.
2) O sistema conduz o usuário para uma interface, na qual todos os dados referentes à Figura poderão ser consultados.

Alterar Figura

1) Após o ponto 1 do Fluxo Alternativo Consultar Figura, o usuário clica no link Editar relacionado à Figura desejada.
2) O sistema conduz o usuário para uma interface de alteração, na qual todos os dados referentes à Figura estarão preenchidos.
3) O Usuário modifica os dados que deseja e clica no botão Editar.

Excluir Figura

1) Após o ponto 2 do Fluxo Básico, o usuário clica no link Remover relacionado à Figura desejada.
2) Caso o sistema verifique que é possível a exclusão, a Figura é excluída e a lista é atualizada. Do contrário, uma mensagem é enviada para o usuário informando que a exclusão não foi possível.

Requisitos Especiais

Não existem.

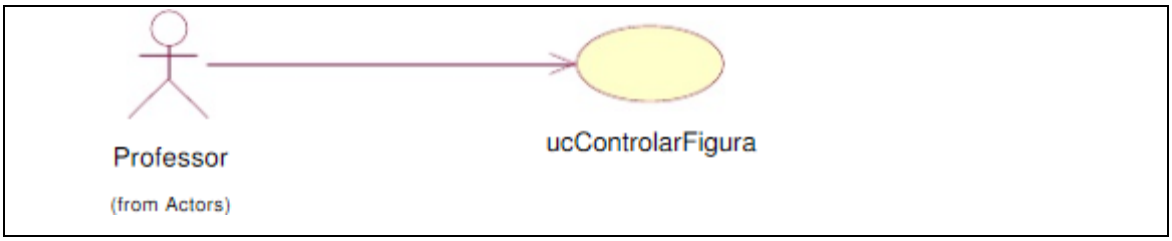
Pré-condições

O usuário deve estar logado no sistema.

Pós-condições

Não existem.

Diagrama de Caso



3.1.3 Prós e Contras da modelagem com Casos de Uso

A modelagem de requisitos funcionais (e mesmo o próprio desenvolvimento de softwares), mediante a especificação de casos de uso, é, atualmente, considerada uma abordagem extremamente adequada, pois facilita a comunicação entre a equipe de projeto e os clientes/usuários, e, ainda, promove a comunicação, o gerenciamento e a condução do desenvolvimento do projeto (RAMOS, 2006)

Uma das principais vantagens de diagramas de casos de uso é a sua capacidade de capturar uma visão ampla da funcionalidade principal do sistema de uma maneira facilmente compreendida por usuários não técnicos. Os diagramas de caso de uso pode se tornar um roteiro centralizado dos cenários de uso do sistema para as pessoas especificarem os requisitos do sistema (DOUGLASS, 2004).

É evidente que uma das vantagens fundamentais da utilização dos casos de uso é a sua simplicidade na escrita de forma natural das especificações, que podem ser entendidas por todos os envolvidos no projeto, inclusive os clientes que, normalmente, não possuem conhecimento técnico.

De acordo com Firesmith (1996), os casos de uso podem ajudar a gerenciar a complexidade de grandes projetos de decompor o problema em funções principais (ou seja, casos de uso), especificando as aplicações a partir da perspectiva dos usuários.

Entretanto, vários problemas têm sido apontados como decorrentes da utilização dos casos de uso. Basicamente, estes problemas resultam da ênfase excessiva no detalhamento do comportamento do sistema, da falta de uma regra objetiva para orientar os analistas na escolha dos níveis de abstração a serem adotados na modelagem, e do insuficiente detalhamento da informação que flui entre o sistema e o seu ambiente (FORTUNA; BORGES, 2005).

A seguir uma visão geral dos principais riscos associados a casos de uso (FIRESMITH, 1996):

- Os casos de uso não são orientados a objetos. Cada caso de uso captura uma importante abstração funcional que pode causar

numerosos problemas com a decomposição funcional o qual era objetivo da tecnologia evitar. Esses problemas incluem:

- A natureza funcional dos casos de uso leva naturalmente à decomposição funcional de um sistema. Cada caso de uso individual envolve diferentes características de vários objetos e classes, e cada objeto individual ou classe é frequentemente envolvido na implementação de vários casos de uso. Portanto, qualquer decomposição baseada em casos de uso espalha as características dos objetos e classes entre os casos de uso individuais. Em projetos grandes, diferentes casos de uso são frequentemente atribuídos a diferentes equipes de desenvolvedores ou diferentes versões e releases. Como os casos de uso não mapeiam um-para-um os objetos e classes, essas equipes podem facilmente criar e códigos múltiplos, redundantes, parciais variantes das mesmas classes, produzindo uma diminuição correspondente na produtividade, reutilização e facilidade de manutenção.
- O modelo de casos de uso e o modelo de objeto pertencem a diferentes paradigmas (ou seja, funcional e orientado a objetos) e, portanto, usam conceitos, terminologia, técnicas e notações diferentes. A estrutura simples do modelo de casos de uso não mapeia claramente a estrutura de rede do modelo de objeto com seus objetos de colaboração e de classes.
- Outro problema potencial com a modelagem de casos de uso é não saber quando parar. Quando se está construindo uma aplicação não trivial, muitas vezes há uma grande quantidade de casos de uso que podem produzir um número praticamente infinito de cenários de uso. Casos de uso demais levam à decomposição funcional e a dispersão dos objetos e classes para os quatro ventos. Muitas vezes, os sistemas e engenheiros de software devem limitar a sua análise para os cenários mais óbvios ou importantes e esperar que a sua análise generalize todos os casos de uso.

  - Embora os casos de uso sejam abstrações funcionais, modelagens de caso de uso no geral não se aplicam ainda a todas as técnicas tradicionais, que são úteis para analisar e projetar abstrações funcionais. A maioria das técnicas atuais não lida facilmente com a existência de ramos e alças na lógica de um caso de uso.
  - Ser criado no mais alto nível de abstração antes de objetos e classes serem identificados fazem os casos de uso ignorarem o encapsulamento de atributos e operações dos objetos. Os casos de uso, portanto, tipicamente ignoraram as questões de modelagem de estado que claramente impactam na aplicabilidade de alguns deles.
- Outro grande problema com a modelagem de caso de uso é a falta de formalidade na definição dos termos de casos de uso, o ator, amplia e usa. Da mesma forma, a especificação de casos de uso nas línguas naturais, oferece um amplo espaço para mal-entendidos. Os casos de uso fornecem uma especificação muito menos formal de suas instâncias (ou seja, cenários de uso individual) do que as classes de objetos. Considerando que a relação de herança entre classes de objetos é bem definida e foi automatizada por compiladores, a "herança" e "delegação" das relações proporcionadas pelas amplia e usa e as associações são menos bem definidas. Embora tudo pareça claro, a tradução de casos de uso em design e em código nos níveis mais baixos de abstração é baseada no entendimento humano informal do que deve ser feito. Isso também causa problemas quando se trata de utilizar casos de uso para a especificação dos testes de aceitação, porque os critérios para passar os testes podem não ser adequadamente definidos.
- Outro grande problema corresponde à arquitetura do subsistema arquetípica que pode resultar de forma cega usando casos de uso.
- Os casos de uso são definidos em termos de interações entre um ou mais atores e o sistema a ser desenvolvido. No entanto, todos os sistemas que não tenham atores podem incluir funcionalidades de significação que não é uma reação à entrada de um ator. Sistemas embarcados podem desempenhar funções importantes, sem controle de

entrada significativa do usuário. Modelagem de casos de uso tradicional parece menos adequada para tais aplicações.

- Finalmente, a utilização de casos de uso como base do desenvolvimento incremental e acompanhamento de projetos tem suas limitações. Baseando incrementos em casos de uso funcionais podemos causar os mesmos problemas com versões baseando-se nas funções principais do sistema. Em vez de construir classes completas, os desenvolvedores tendem a criar variantes parciais que requerem mais iterações de construção para o desenvolvimento do que é necessário. Por sua vez, isso vai aumentar desnecessariamente os custos de manutenção.

3.2 SADT

A sigla SADT vem do inglês *Structured Analysis and Design Technique*, o que significa “Análise Estruturada e Técnica Design”. Esta técnica foi desenvolvida pela SofTech entre 1972 e 1975. Abrange a análise de requisitos, o projeto e a documentação de especificações, visando uma melhor comunicação entre os analistas, desenvolvedores e usuários (KÜNDIG; BÜHRER; DÄHLER, 1987).

SADT é uma linguagem gráfica para descrever sistemas, em conjunto com uma metodologia para a produção de tais descrições. Ele pode ser aplicado em uma variedade de sistemas (baseado em computador ou de outro modo), e é de uso particular durante a análise dos requisitos de design e software e trabalha com o princípio de dividir e conquistar (BIRREL; OULD, 1985).

Analisando o histórico do SADT, fica claro que a técnica trazia algumas características revolucionárias que auxiliavam na descrição para o desenvolvimento de softwares complexos. As características mais marcantes são a decomposição funcional e sua representação esquemática simples.

SADT é um método que enfatiza em seu contexto os aspectos da organização onde “viverá” o sistema sendo desenvolvido, procurando assim maximizar a captura de requisitos importantes para o desenvolvimento de um software com certo grau de completeza (RIBEIRO, 1991).

Dessa forma, além dos requisitos funcionais, o método SADT permite também a formalização dos requisitos não-funcionais, comportando o desenho de uma solução mais completa e compatível com os problemas apresentados.

A modelagem com SADT fornece métodos para (BIRREL; OULD, 1985):

- Manuseio de Problemas complexos com uma boa através de uma rigorosa decomposição.
- Definir e gerenciar os papéis e relacionamentos do pessoal envolvido.
- Registar e apresentar os resultados de entrevistas, análise e desing.
- Garantia da qualidade (por exemplo, precisão e integridade) por meio de revisão.

Esse método é composto de duas partes:

- SA: *Structured Analysis*, a análise estruturada.
- DT: *Design Technique*, a técnica de projeto.

Nesta seção é apresentado um estudo básico focado no SA (*Structured Analysis)*, ou seja, um estudo simplificado da linguagem gráfica de diagramação.

3.2.1 O método SADT

O método SADT concentra-se no fluxo de dados e implica no refinamento gradual dos chamados Diagramas-SADT que são hierarquicamente ordenadas. Na sua definição original, existe uma dualidade entre os chamados *actigramas* e *datagramas*, modelando o fluxo de dados em dois modos diferentes que representam diferentes pontos de vistas do sistema (KÜNDIG; BÜHRER; DÄHLER, 1987):

- *Actigramas* identificam as funções como elementos centrais da descrição e de dados fornecendo, por exemplo, a entrada ou saída para as funções.
- *Datagramas* identificam dados como elementos centrais da descrição e funções fornecendo, por exemplo, a entrada ou de saída para os dados.

A redundância torna possível provar a consistência, isto é, pode-se verificar se a função de *actigrama* é compreendida no *datagrama*, e vice-versa.

O SADT possui um *actigrama* que representa atividades e um *datagrama* que é usado para representar dados. Esses dois tipos de diagrama, utilizam a estrutura de caixa e setas, conforme representado na figura abaixo:

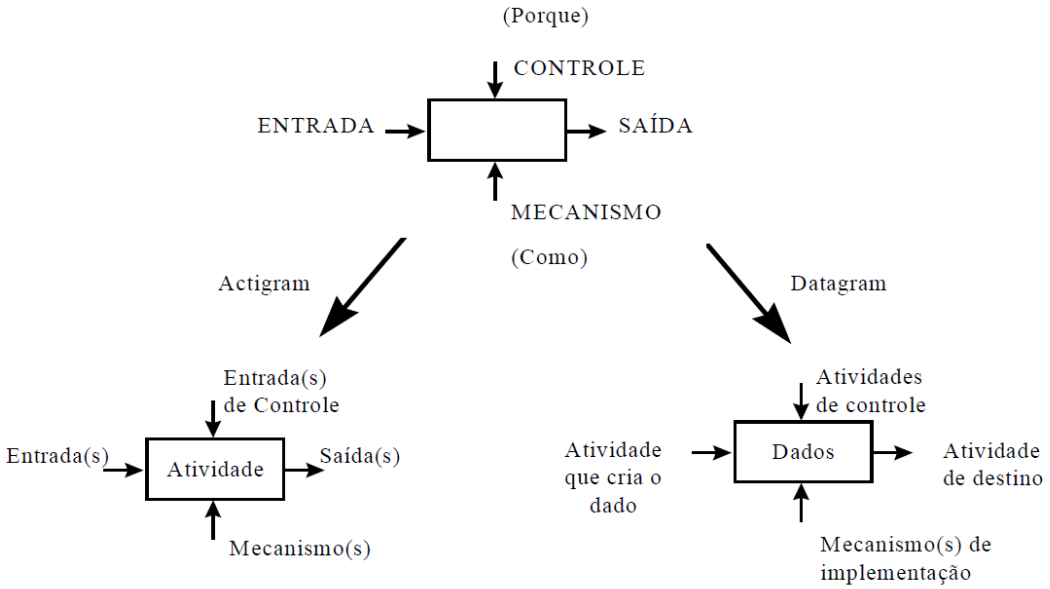


Figura 8. Notação SADT – Actigrama e DataGrama

(NETO; SANTOS, 2012)

Cada modelo em SADT consiste em uma hierarquia de diagramas, descrevendo o sistema de um particular ponto de vista. Assim um mesmo sistema, pode ser representado por um número de modelos correspondentes a diferentes visões (RIBEIRO, 1991). Dessa forma, uma hierarquia de diagramas descreve o sistema de um ponto de vista específico com um único propósito.

3.2.1.1 Validação do modelo SADT

A validação do modelo SADT faz parte do método dessa técnica de modelagem.

O método SADT estabelece títulos e funções para a equipe de desenvolvimento denominado de ciclo autor/leitor. O objetivo desse ciclo é promover a revisão do modelo, o que é feito de maneira informal, validando assim a especificação contida no diagrama. Tal revisão consegue atingir também os requisitos não-funcionais (RIBEIRO, 1991).

A etapa de validação é como um ciclo de inspeção, que permite a revisão, críticas e correções do modelo apresentado, a fim de diminuir erros de programação ou o desenho de uma solução que não resolva o problema, evitando custos desnecessários de correções.

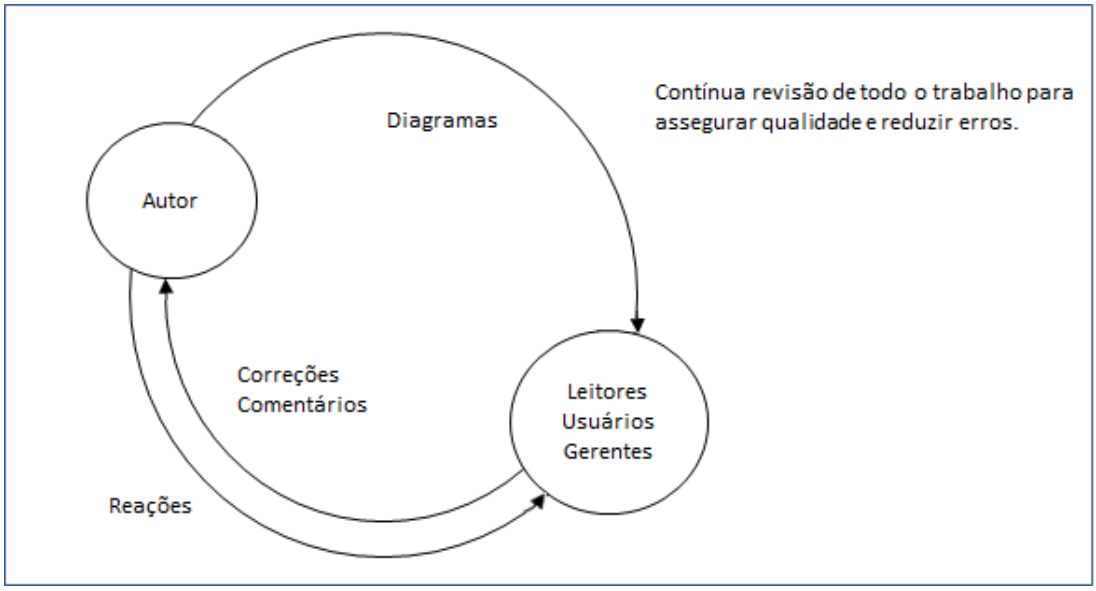


Figura 9. Ciclo Autor/Leitor do SADT

(RIBEIRO, 1991)

3.2.2 A linguagem gráfica SADT

A notação básica do SADT é composta por uma caixa, que representa uma atividade ou um dado, além disso, existe um conjunto de quatro setas, conforme apresentado na figura 8, onde cada uma destas setas tem um significado:

- A seta da esquerda corresponde aos dados de entrada
- A seta da direita corresponde aos dados de saída.
- A seta superior, que toca o topo da caixa, corresponde aos controles e restrições da atividade.
- A seta inferior, que toca a parte de baixo da caixa, corresponde aos mecanismos.

As setas de entrada em uma caixa representam para o caso de uma atividade os dados disponíveis para leitura ou consumo, enquanto que as setas de saída representam o dado que é produzido pela atividade. A seta de controle governa a maneira pela qual a transformação ocorre. Em geral a atividade tem por objetivo a transformação de um dado de entrada em um dado de saída. A seta mecanismo é muito poderosa, sendo uma importante parte da sintaxe do SADT (RIBEIRO, 1991).

Um modelo SADT possui uma sequência de diagramas, que conforme são refinados, apresentam uma quantidade maior de detalhes. Logo, o nível mais alto apresenta o sistema como um todo. O nível de detalhamento é limitado pelo ponto

de vista. Os diagramas são utilizados também como a representação de uma especificação de requisitos do sistema.

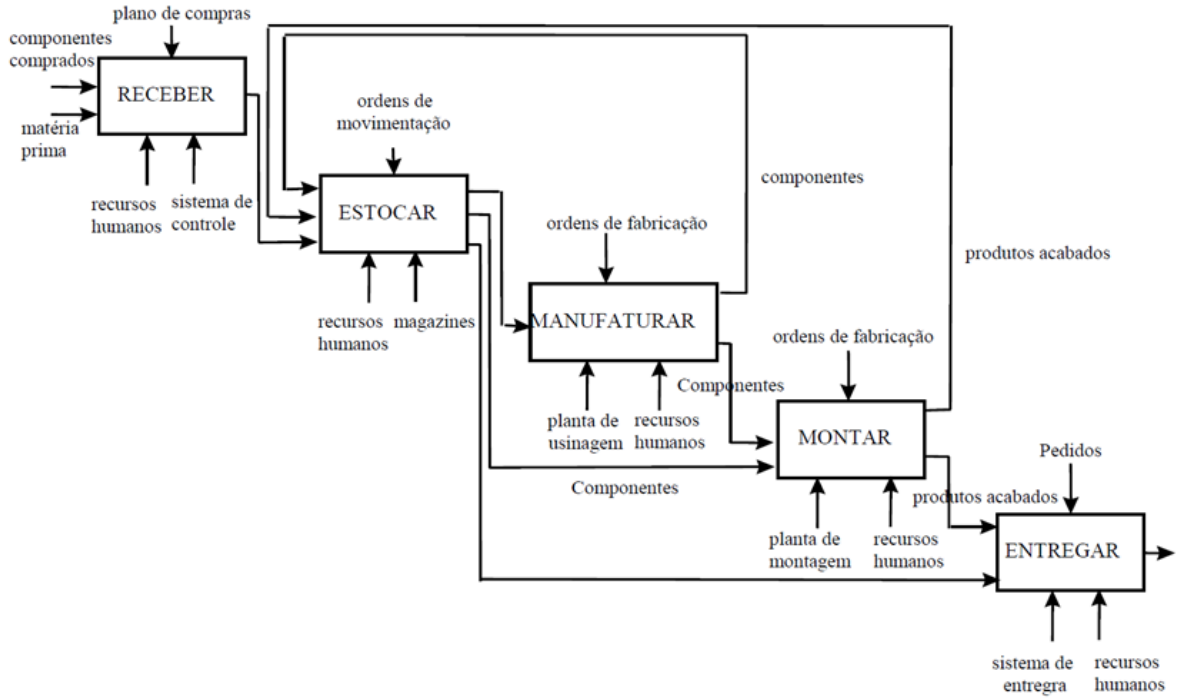


Figura 10. Exemplo prático do modelo SADT

(NETO; SANTOS, 2012)

Um modelo SADT completo deve conter (RIBEIRO, 1991):

- *Actigramas* (Diagramas do modelo Funcional);
- *Datagramas* (Diagramas do modelo de Informação);
- FEOs ("*For Exposition Only*" - Diagramas utilizados para Comentários);
- Texto (Acompanhando cada diagrama com explicações em linguagem natural);
- Índice de Nodos (cada diagrama é um nodo da árvore que representa o desenvolvimento da solução);

3.2.3 Prós e Contras da modelagem com SADT

A modelagem SADT permite o desenho de soluções de problemas complexos através de sua decomposição e níveis de detalhamento de forma organizada.

Através do processo de validação que faz parte do método SADT é possível ter garantia da qualidade por meio de revisão e facilitar a comunicação entre os envolvidos.

SADT trabalha com modelos. Cada modelo consiste de uma hierarquia de diagramas que descrevem um sistema a partir de um ponto de vista específico.

Dessa maneira, um sistema pode ser representado por um distinto número de modelos correspondentes a diferentes pontos de vista. Tais modelos podem, por exemplo, ser do ponto de vista do operador, do ponto de vista industrial ou de vendas. Assim, o uso de diagramas leva em conta o nível de abstração do sistema e o ponto de vista sobre o qual o analista deseja apresentar o sistema.

O modelo utiliza notações e padrões fáceis de aprender, que não requerem conhecimentos de software ou análise, facilitando a comunicação com os usuários finais e outras partes da empresa, como por exemplo, a equipe de vendas.

Isso ajuda a definir de maneira mais clara os objetivos do modelo, como também as condições em que foi criado.

A principal desvantagem do SADT é a sua riqueza: um diagrama SADT contém tanta informação, que pode ser difícil para os usuários de apreciá-lo como um modelo adequado e correto para os seus sistemas (AKTAS, 1987).

Logo, apesar do SADT permitir e especificar a utilização da linguagem natural em seus diagramas e anotações, a sua complexidade e nível de detalhamento podem atrapalhar o entendimento do modelo.

São apresentadas ainda, problemas com a modelagem relacionados ao tempo de criação e manutenção dos diagramas (LARA, 1992):

- Os procedimentos de diagramação podem levar muito tempo e diminuir a flexibilidade de comunicação.
- Manter a documentação em dia requer muito tempo e bastante disciplina.

3.3 Redes de Preti

As redes de Petri foram criadas em 1962 por Carl Adam Petri como ferramenta para modelar e analisar processos. Ao longo dos anos, o modelo proposto por Petri tem sido estendido de várias formas (AALST; HEE, 2009).

Rede de Petri é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática (FRANCÊS, 2003). Esta técnica é uma ferramenta gráfica e matemática que se adapta bem a um grande número de aplicações em que as notações de eventos e de evoluções simultâneas são importantes (CARDOSO; VALETTE, 1997).

Redes de Petri foram projetadas inicialmente para modelar sistemas com interação de componentes simultâneos. Mas pode-se enfatizar o uso desta técnica, principalmente durante a concepção, onde pode ser aplicada com bons resultados na definição de sistemas.

Umas das características mais importantes desta técnica é o fato dela ser formal. De acordo com Moura (2001), um modelo formal é um método de desenvolvimento de software através do qual se pode definir precisamente um sistema e desenvolver implementações garantidamente corretas em relação a esta definição.

3.3.1 Representação gráfica das Redes de Petri – Conceitos Básicos

Uma rede de Petri básica é composta por:

- Lugares: representada por um círculo, correspondem ao estado do sistema.
- Transições: representada por uma barra, correspondem às transições responsáveis pela mudança do estado.

Esses dois componentes, lugares e transições, estão ligados através dos chamados arcos dirigidos. Os arcos indicam para as transições os lugares que elas operam.

Existem ainda as marcas, que também são chamadas de *tokens,* são representadas por círculos menores pretos localizados dentro dos lugares e correspondem ao estado do lugar e as transições que permitem transformações.

As marcas dos lugares podem ser abstraídas, representando um estado de determinado evento, por exemplo, documento disponível (lugar com marca) ou documento indisponível (lugar sem marca) (PÁDUA et al, 2004).

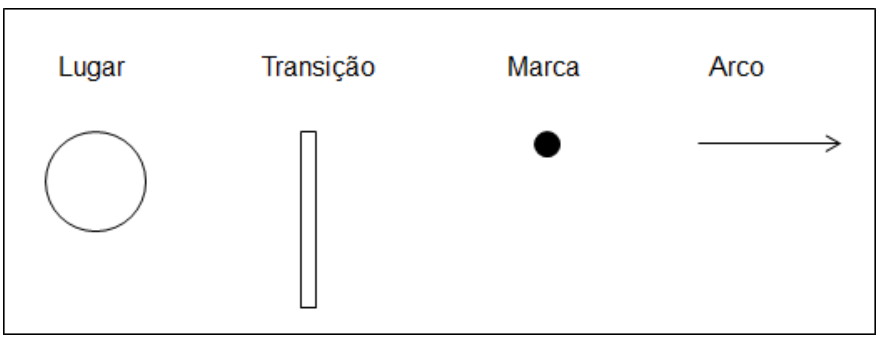


Figura 11. Elementos Básicos da Rede de Petri

Como exemplo prático, Guimarães (2003) define a rede de Petri como uma quádrupla (P, T, F, M_o) onde P é um conjunto de lugares, T um conjunto de transições, F um conjunto de arcos e M_o a marcação inicial dos lugares. Uma rede de Petri é representada graficamente na Figura 12.

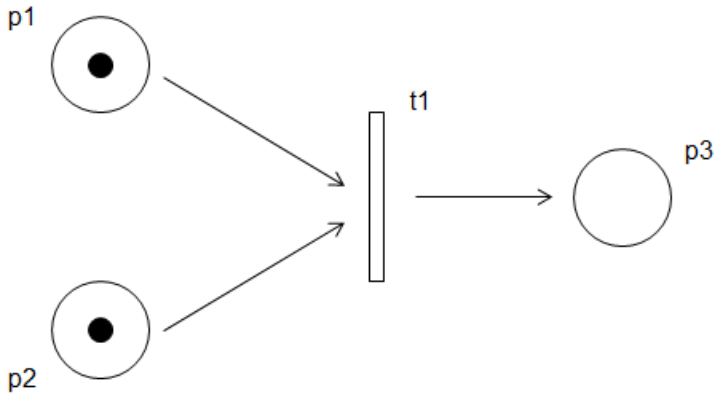


Figura 12. Exemplo de uma Rede de Petri

A rede de Petri apresentada na figura 12 é representada matematicamente por:

({ p_1, p_2, p_3 },

{ t_1 },

{ (p_1, t_1), (p_2, t_1), (t_1, p_3) },

{ (p_1, 1), (p_2, 1), (p_3, 0) })

Onde cada uma das linhas representa:

- ({ p_1, p_2, p_3 }: Os lugares da rede.
- { t_1 }: As transições da rede.
- { (p_1, t_1), (p_2, t_1), (t_1, p_3) }: o relacionamento entre os lugares e transições e transições e lugares.
- { (p_1, 1), (p_2, 1), (p_3, 0) }): as marcas ou *tokens* presentes em cada lugar.

Redes de Petri são executadas quando as marcas são retiradas de alguns lugares e colocadas em outros. Quando a rede é executada, as transições retiram as marcas de entrada e as colocam na saída. Assim, se a rede da Figura 12 for executada, teremos como resultado a rede da Figura 13.

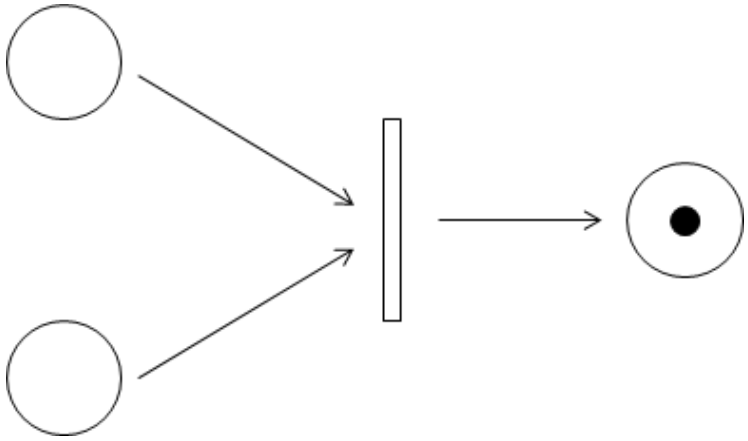


Figura 13. Resultado da execução da rede de Petri da Figura 12.

3.3.2 Modelagem com Redes de Petri

Quando modelamos um sistema através de uma rede de Petri, estamos necessariamente criando uma interpretação da rede, no sentido de dar significado a representação gráfica. Essa interpretação ou significação que efetua a ligação do modelo abstrato que qualquer rede de Petri representa, com o sistema concreto que se pretende modelar. Por exemplo, uma possível interpretação da rede da Figura 14, é a modelação de um sistema produtor-consumidor.

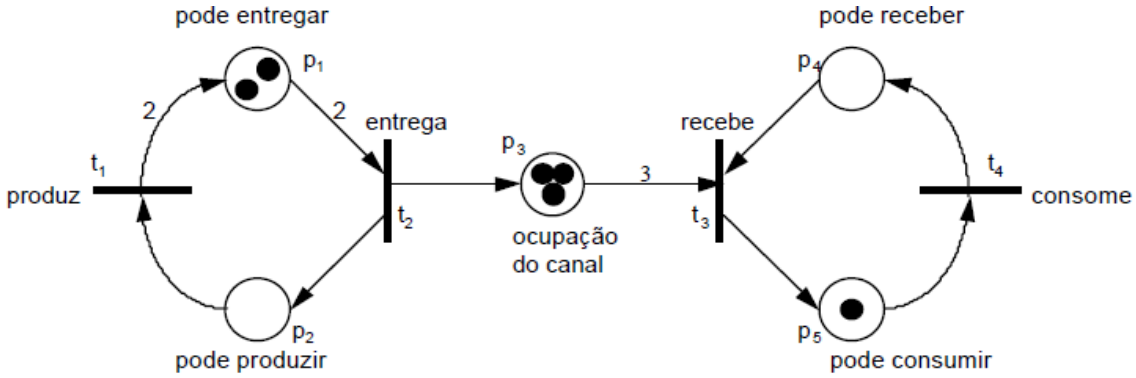


Figura 14 – Modelagem Rede de Petri com interpretação

(Barros, 2001)

Conforme interpretação de Barros (2001), na Figura 14 é perceptível que os produtos são produzidos aos pares e que para cada um destes pares é entregue uma unidade. Por outro lado, o consumidor (modelado pelos lugares p_4 e p_5 e pelas transições t_3 e t_4) necessita de receber três unidades e, após as consumir, antes de poder receber mais unidades.

Torna-se imprescindível o uso da interpretação ou significado para o desenho de modelos utilizando as redes de Petri.

As redes de Petri possuem diversos modelos que as representam. Através dos diferentes tipos de modelos é possível a visualização de conceitos, relações, ações, condições, conflitos, fluxos e outros requisitos do sistema.

Na próxima seção apresentaremos alguns desses modelos.

3.3.2.1 Máquinas de Estados Finitos

Máquinas de Estado Finito são chamadas assim porque tem um número limitado de respostas, ela é especialmente útil para modelar o comportamento de sistemas reativos, os quais são essencialmente dirigidos a eventos e dominados por controles. Além disso, possuem uma gama de aplicação bastante grande e genérica, podendo ser utilizadas na modelagem de vários tipos de sistema (WIKISPACES, 2009).

Com essa técnica, cada máquina de protocolo (isto é, o transmissor ou o receptor) está sempre em um estado específico a cada instante. Seu estado consiste em todos os valores de suas variáveis, inclusive o contador de programa (TANENBAUM, 2003).

A representação gráfica básica deste modelo pode ser vista na figura 15.

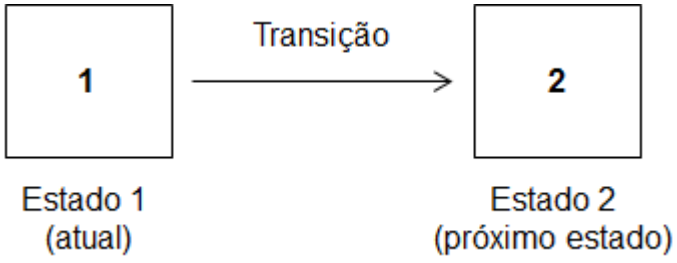


Figura 15 – Representação Gráfica Maquinas de Estados Finitos

Considerando o exemplo prático de Guimarães (2003), referente a uma máquina de venda automática de doces que aceita moedas de 5 e 10 centavos. A máquina vende doces de 15 e 20 centavos. Assuma que não mais do que 20 centavos podem ser colocados na máquina. A modelagem está na Figura 16.

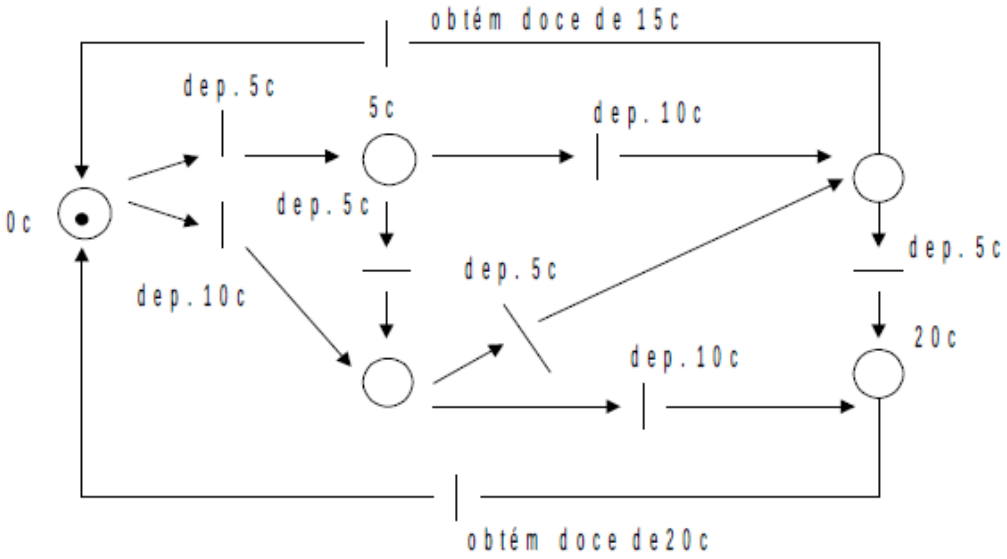


Figura 16 – Exemplo Maquinas de Estados Finitos

(Guimarães, 2003)

3.3.2.2 Atividades Paralelas

É possível representar uma atividade paralela em uma rede de Petri sempre que duas transições diferentes e independentes são capazes de disparar ao mesmo tempo, conforme representado na figura 17, onde as transações t_2 e t_3 são consideradas paralelas.

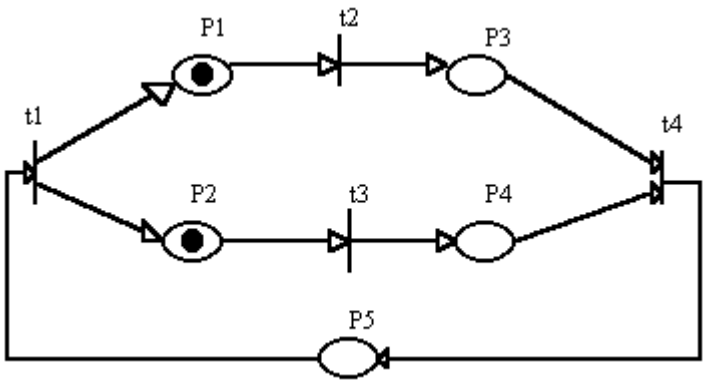


Figura 17 – Exemplo de Atividades Paralelas

(PALOMINO, 1995)

Cardoso e Valette (1997) apresentam dois tipos de parelelismos:

- Pseudo-paralelismo: o paralelismo é apenas aparente e os eventos, mesmo independentes, nunca serão simultâneos. Eles serão ordenados por um relógio comum. É o caso de várias tarefas informáticas sendo executadas num único processador. Este executa somente uma instrução por vez.

- Paralelismo verdadeiro: os eventos podem ocorrer simultaneamente. Isto significa que não existe uma escala de tempo comum suficientemente precisa para determinar qual evento precedeu o outro. Ocorre quando várias tarefas informáticas são executadas num computador paralelo, com um processador alocado para cada tarefa independente.

3.3.2.3 Sincronização

Em redes de Petri sincronizadas, um evento é associado a cada transição e o disparo da transição somente ocorrerá (MONTEZANO, 2009):

1. Se a transição estiver habilitada;
2. Quando o evento associado ocorrer.

Ainda é possível afirmar que a sincronização modela a junção entre as atividades concorrentes.

A figura 18 representa claramente uma rede de Petri sincronizada.

O item “a)” representa a sincronização recíproca de duas evoluções. Na parte esquerda, a posição P_1 está marcada, mas não pode evoluir enquanto a posição P_3 não estiver marcada também. A partir do momento em que ambas as posições estejam marcadas a transição t_1 passa a estar disponibilizada e poderá ser disparada. O que foi exposto aplica-se ao caso de apenas P_3 estar marcada

O item “b)” ilustra uma situação um pouco diferente, mas também de sincronização. Neste exemplo, a evolução da parte esquerda é independente da parte direita. O mesmo não se pode dizer da parte direita. A transição t_2 só pode ser disparada após o disparo da transição t_1. A posição P_1 memoriza a autorização de disparo da transição t_2 (sincronização por semáforo).

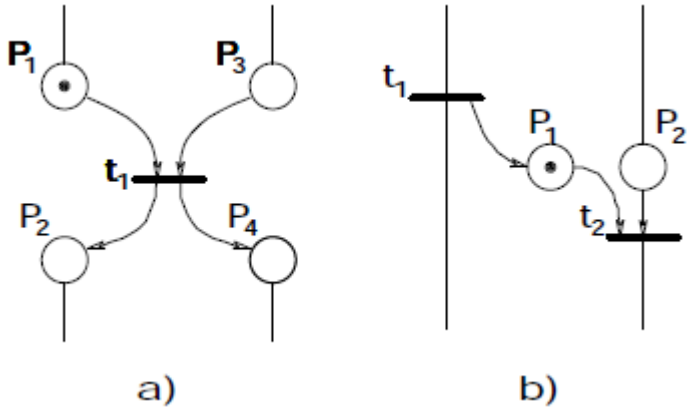


Figura 18 – Exemplo de rede com Sincronização

(TOVAR, 2007)

3.3.2.4 Partilhamento

Sem dúvida, a utilização de recursos, e principalmente o seu partilhamento, é um dos pontos mais importantes na modelagem de um sistema. Uma vez que um recurso está ocupado com uma atividade, não pode estar disponível para outras. Embora a ideia seja trivial, a verificação nem sempre é. A modelagem do partilhamento de recursos é fundamental para a representação correta de um sistema (CARDOSO; VALETTE, 1997).

Um exemplo prático onde o partilhamento pode ser utilizado é o controle de trens. Pensando de forma simplificada, para não haver colisões, só pode existir um trem por estação. Para entrar em uma estação, é necessário verificar se ela está livre: a estação é, portanto, considerada um recurso repartido entre os trens.

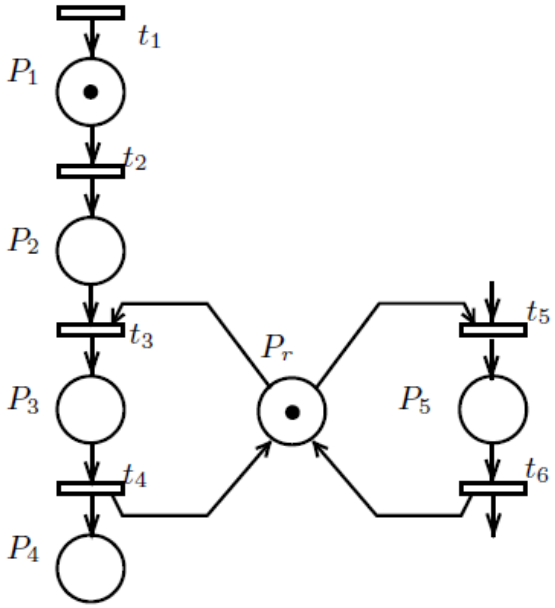


Figura 19 – Exemplo de rede com Partilhamento

(CARDOSO; VALETTE, 1997)

Observando a figura 19, considere que, após uma atividade P_1, é preciso executar uma operação que necessita a utilização de um recurso r representado pelo lugar Pr na figura. Pr marcado corresponde ao estado parcial recurso disponível. A transição t_3 exprime a tomada do recurso e o início da fase P_3.

Para modelar separadamente o final da atividade P_1 e o início de P3, é preciso introduzir a transição t_2 associada ao fim da atividade P_1. O lugar P2 corresponde à

espera do recurso associado ao lugar Pr, se este não está disponível. O lugar P_2 permite representar o estado do sistema em que a atividade P_1 já foi executada, esperando que o recurso se libere para executar a atividade P_3.

A ausência de t_2 e P_2 nesta rede modelaria um comportamento diferente. Como t_3 só pode disparar se o lugar Pr está marcado, a espera do recurso bloquearia, neste caso, o fim da atividade P_1.

3.3.3 Prós e Contras da modelagem com Redes de Petri

As redes de Petri são baseadas no método formal. O uso de conceitos formais apresenta grande número de vantagens (AALST; HEE, 2009):

- Obriga a uma definição precisa, prevenindo ambiguidades, incertezas e contradições.
- O formalismo pode ser usado para discutir processos, sendo possível estabelecer padrões.
- Permite o uso de técnicas analíticas (aquelas para analisar o desempenho, por exemplo, bem como aquelas para verificar propriedades lógicas).

Além disso, a modelagem com redes de Petri permitem o desenho de softwares mais complexos de forma precisa e garantem que este cumpre os requisitos pedidos. Com a utilização de métodos matemáticos é possível provar que a implementação corresponde ao que foi especificado.

Entretanto, existem algumas desvantagens as quais levam as redes de Petri a não serem utilizadas em larga escala:

- São necessários conhecimento e treinamento técnico da equipe para especificação e entendimento do que foi modelado.
- Os clientes e usuários tendem a evitar essa técnica por não serem familiarizadas com a mesma e não as financiam.
- Algumas classes de software são difíceis de especificar (FIGUEIREDO et al, 2002):
    - Sistemas interativos
    - Sistemas concorrentes

Ainda são apresentadas outras desvantagens em relação ao modelo (SANTANA; SANTANA, 2011):

- Um lugar não pode ser subdividido em sub lugares, o que pode levar à explosão do número de lugares e transições do modelo;
- Poucas ferramentas implementam extensões hierárquicas, que possibilitam uma maior compactação do modelo.

3.4 SREM

SREM é uma sigla para *Software Requirements Engineering Methodology* que traduzindo para o português ficaria Metodologia de Engenharia de Requisitos de Software. SREM foi originalmente desenvolvida como um auxílio automatizado na definição de requisitos de software para o Sistema de Armas de Defesa de Mísseis Balísticos dos Estados Unidos. Desde então, foi comercializado pela TRW nos Estados Unidos e tem sido usado por uma série de organizações (BIRREL; OULD, 1985).

A abordagem SREM é representada pelas seguintes características (ALFORD, 1977):

- O uso de uma notação gráfica executável para descrever o comportamento do sistema e do software (por exemplo, RNETs e FNETs).
- O uso de um usuário extensível-elemento-relação-atributo (ERA) linguagem para descrever os diversos requisitos do sistema e desenhar suas características, incluindo a rastreabilidade e decisões.
- O uso de ferramentas automatizadas para aceitar as informações acima, verificação do desempenho estático e dinâmico da consistência-completude, validar as especificações mediante a execução delas, e gerar os documentos de especificação necessários.
- A definição de um método para usar a linguagem e as ferramentas descritas como uma sequência de passos, cada um dos quais adiciona um subconjunto específico do ERA e informações executáveis e, em seguida, utilizar as ferramentas para verificar um subconjunto das condições consistência-completude.
- A intenção de proporcionar uma transição harmoniosa de requisitos de sistema e design, através de requisitos de software, design e testes.

A figura 20 apresenta uma visão geral do método SREM o qual começa com a documentação de especificação do sistema e fornece os requisitos de processamento de dados.

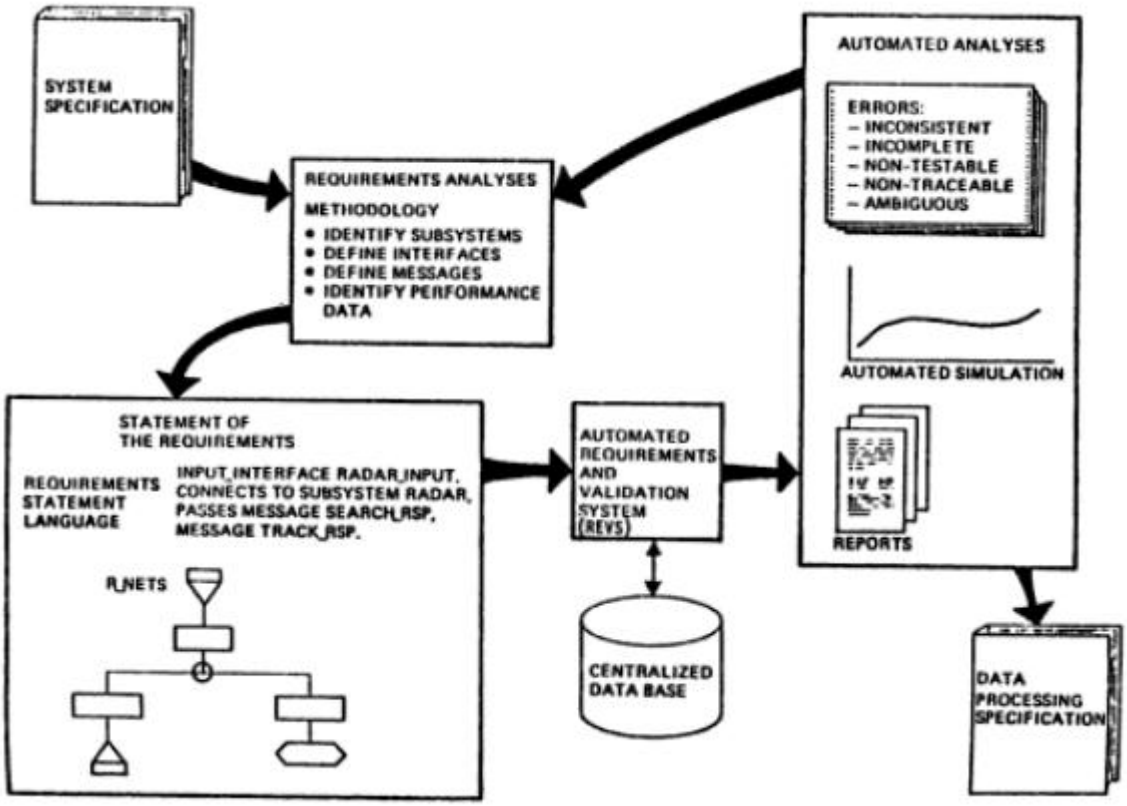


Figura 20 – Visão Geral SREM

(ANDERSON; DORFMAN, 1991)

SREM pode ser dividido em uma linguagem processável para estabelecer de requisitos (RSL: *Requirements Statement Language*) e um conjunto de ferramentas (REVS: *Requirements Engineering and Validation Systems*) para traduzir a linguagem, realizar a execução e validação da consistência automaticamente e gerar documentações, procedimentos e simulações.

3.4.1 RSL: Estabelecimento dos Requisitos

RSL é uma linguagem processável para especificar requisitos que estabelece processamentos através de um conjunto de entradas e saídas. Os caminhos e passos do processamento são representados de forma gráfica através das R-nets, como apresentado na figura 21.

R-nets são utilizadas para estruturar a descrição do processo. A R-net é traduzida para uma linguagem estruturada em inglês usando o padrão de declaração requisitos para análise através dos REVS. Os requisitos são então armazenados na base de dados central e são analisados pelos REVS. Por fim, o relatório de resultados aponta os erros de inconsistência, incompletude, testabilidade e rastreabilidade (ANDERSON; DORFMAN, 1991).

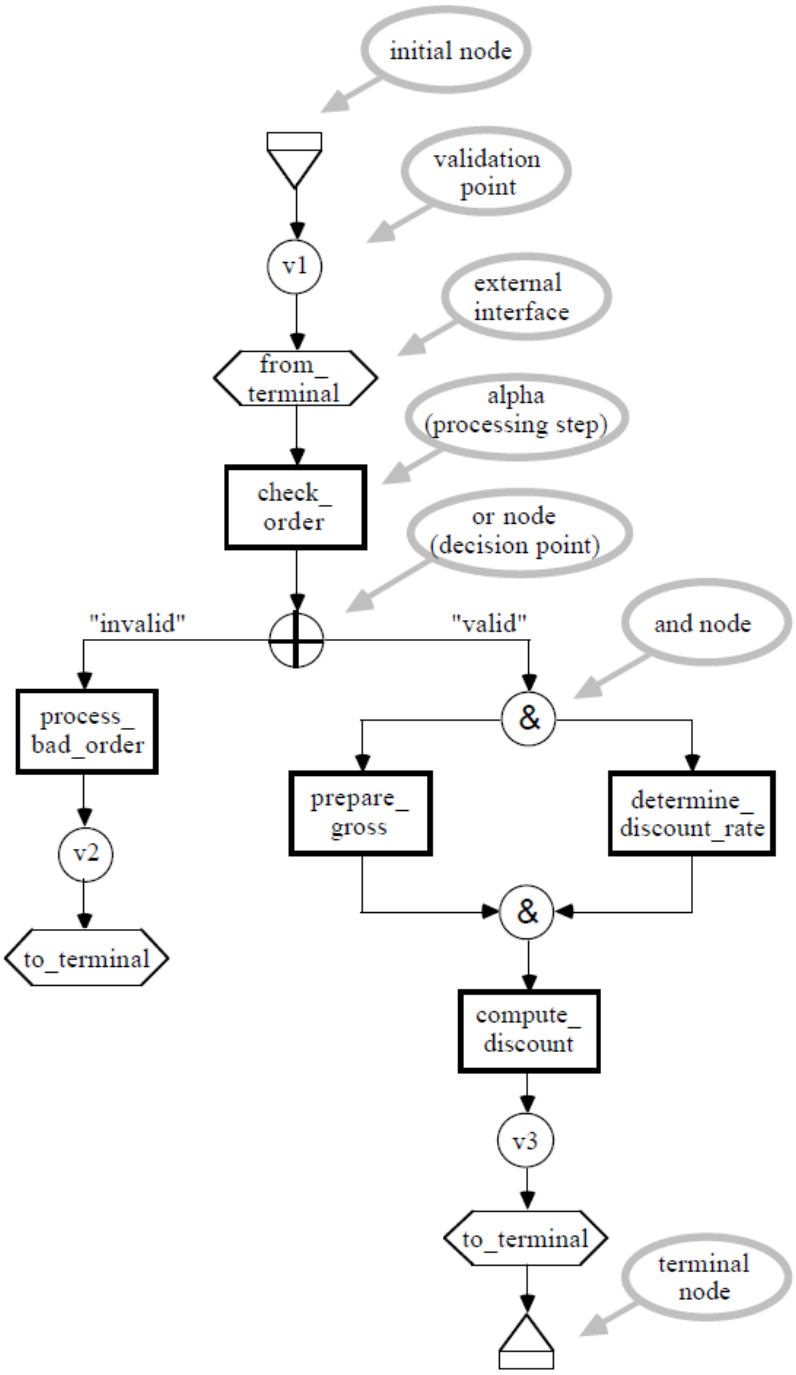


Figura 21 – Representação gráfica RSL com R-net

(TSE; PONG, 1989)

RSL tem quatro conceitos primitivos (CHAAR, 1987):

- Elementos são usados para nomear objetos;
- Atributos são usados para descrever as características dos elementos;
- Relacionamentos são usados para descrever as relações binárias entre elementos;
- Estruturas são usadas para descrever o fluxo de controle de um sistema.

A linguagem RSL foi criada para ser uma forma de estabelecer requisitos processáveis, mas de certa maneira utilizando linguagem natural, embora tendo que manter um nível de padrão e formalidade suficientes para serem interpretadas pelos REVS. A figura 22 ilustra as declarações RSL relacionadas com as R-nets e seus dados, além de apresentar o tipo de inglês utilizado no RSL.

```
INPUT INTERFACE: MULTI.
  DESCRIPTION: 3 TEMPERATURES 3 PRESSURES 2 SWITCHES.
  ENABLES: R NET MONITOR-ENGINES.
  PASSES: MESSAGE ENGINE-DATA.
  DOCUMENTED BY
   SOURCE: TRW-CLASS_NOTES.
  COMPLETENESS: CHANGEABLE.

MESSAGE: ENGINE-DATA.
  MADE BY
   DATA: TEMPERATURE
   DATA: PRESSURES
   DATA: SWITCHES.

DATA: TEMPERATURE.
  INCLUDES
   DATA: INTAKE TEMP
   DATA: COMPRESSION TEMP
   DATA: EXHAUST TEMP.
```

Figura 22 – Exemplo de declarações RSL relacionadas ao R-NET e seus dados

(BOEH, 1984)

3.4.2 REVS: Engenharia de Requisitos e Sistemas de Validação

REVS é formado por um conjunto de ferramentas que permitem processar o RSL para efetuar a análise do banco de dados e verificar a consistência e integridade dos requisitos, apresentando uma análise automática das incoerências encontradas.

REVS também é capaz de efetuar a simulação estática e dinâmica do sistema. Esta informação é sempre realimentada para que o processo seja repetido até que se torne satisfatório.

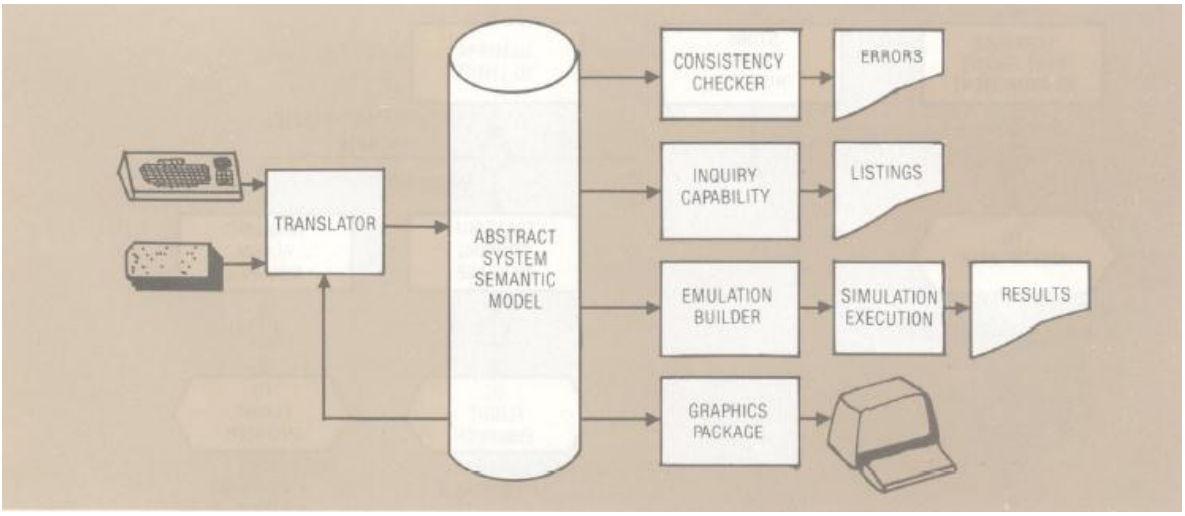


Figura 23 – REVS: Fluxo de informação na Engenharia de Requisitos e sistema de validação

(BOEH, 1984)

REVS é constituído por três componentes principais (CHAAR, 1987):

1. Um tradutor para RSL.
2. Um banco de dados centralizado, o modelo semântico resumido do sistema (ASSM).
3. Um conjunto de ferramentas automatizadas de processamento de informação na ASSM, e, para a geração de simulações funcionais e de análise para permitir a avaliação das características de desempenho de diferentes configurações do sistema.

3.4.3 Prós e Contras da modelagem com SREM

A formalidade da linguagem de especificação utilizada pelo método SREM, o RSL, permite uma descrição dos requisitos mais precisa e não ambígua para definição de softwares. Como resultado, o custo de desenvolvimento e manutenção do software é significativamente diminuído.

Embora as partes do SREM possam ser especificadas e aplicadas manualmente, tal uso carece de alguns dos principais benefícios da combinação do SREM/REVS (BIRREL; OULD, 1985):

- Rastreabilidade automatizada das declarações RSL para os requisitos do sistema de origem.
- Especificação e exibição do trajeto do fluxo de gráficos interativos.
- Verificação automatizada da Integridade e consistência.
- Documentação automatizada dos requisitos.
- Geração automatizada de simulações.

Apesar do custo de desenvolvimento ser diminuído com o uso do SREM, o custo e o tempo de análise é relativamente aumentado. SREM normalmente apresenta um melhor custo-benefício se utilizado na especificação de sistemas grandes, integrados e de tempo real.

De acordo com Chen (1988), a metodologia é mais adequada para o controle intensivo do sistema (por exemplo, controle de um míssil), em vez de dados intensivos do sistema (por exemplo, processamento de folha de pagamento).

Além disso, é necessário treinamento para a especificação correta utilizando a linguagem RSL. Apesar de não ambígua pelo fato de utilizar um método formal, é por esse mesmo motivo que a descrição com RSL dificilmente é compreendida pelos usuários e clientes, que relutam em aceitar essas especificações como parte de um contrato.

3.5 User Stories

Entre as técnicas apresentadas, *User Stories* é a mais atual e a única desenvolvida notadamente para análise de requisitos em sistemas geridos segundo metodologias ágeis.

O conceito de *User Stories* foi introduzido pela primeira vez ao desenvolvimento de software com a publicação do livro *eXtreme Programming* de Kents Beck's em 1999. Este conceito foi posteriormente desenvolvido e alargado aos outros métodos ágeis como Scrum (EOCHA; CONBOY, 2010).

Uma *User Story* descreve uma funcionalidade que será valiosa para um usuário ou comprador de um sistema ou software. Estórias de usuários são compostos de três aspectos (COHN, 2004), conhecidos com *The 3 C's (Card, Conversation* e *Confirmation)*:

- ***Card*:** Uma descrição escrita da estória usada para planejamento e como um lembrete.

- ***Conversation*:** Conversas sobre a estória que servem para concretizar os detalhes da estória.
- ***Confirmation*:** Testes ou critérios de aceitação que transmitem e detalhes do documento e que podem ser utilizados para determinar quando uma estória é completa.

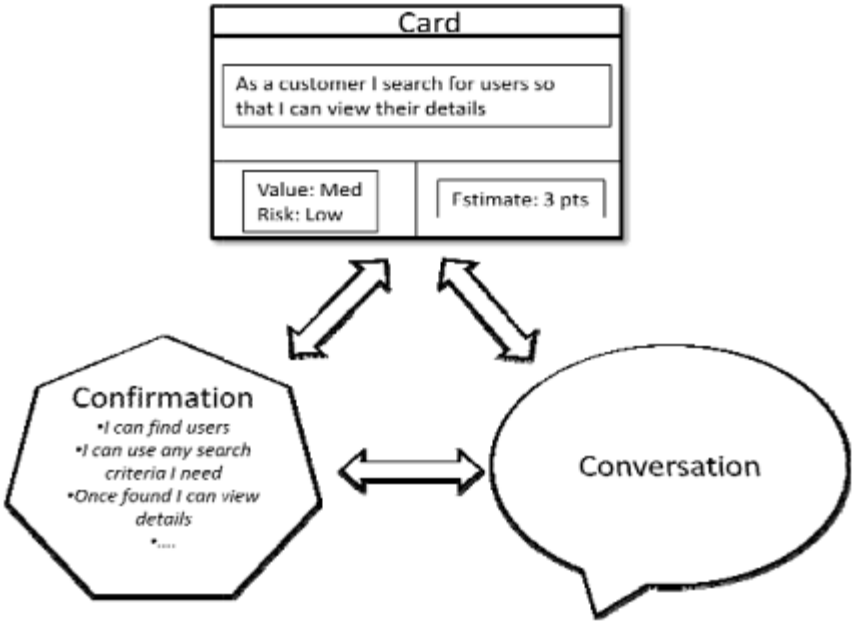


Figura 24 – Elementos de uma User Story

(COHN, 2004)

Os *User Stories* surgiram como uma alternativa para especificação de requisitos para os métodos ágeis, que exigem uma rápida adaptação às mudanças. Existe uma comparação e até uma confusão entre esse método e os casos de uso, pode-se considerar a grande diferença entre esta técnica para o modelo de casos de uso o nível de detalhamento utilizado. Enquanto os casos de uso são abundantemente detalhados, podendo conter regras de negócios, mensagens e tem como foco as interações do usuário com o sistema, o *User Story* foca no objetivo do usuário na utilização do sistema e desta forma não se limitam a descrever uma interface de usuário.

É preciso que fique claro, que essa técnica é recomendável para ser utilizada em projetos que utilizem metodologias ágeis e/ou iterativas, os quais recomendam iterações de duas ou no máximo quatro semanas. Isso porque grande parte do *User Stories* é baseada em conversas, o que perde todo o sentido de ser utilizado em, por

exemplo, um modelo do estilo cascata, no qual todo o levantamento de requisitos do sistema a ser entregue é realizado de uma vez só.

3.5.1 Escrevendo User Stories

Como apresentado, *User Stories* são curtas e possuem uma simples descrição de uma funcionalidade que possui valor para o usuário ou comprador do sistema. Essas estórias podem ser escritas em um pequeno pedaço de papel ou nos tradicionais cartões utilizados com este propósito. O cartão também pode capturar as estimativas iniciais do valor da estória para o cliente, o custo da implementação da mesma (em formas de pontos, por exemplo) e o risco.

Não há uma regra específica de como escrever essas estórias, mas existe um padrão comumente utilizado por equipes ágeis:

"Como [**papel/ator**] eu quero/preciso de [**ação**] para que [**resultado/função**]"

Um exemplo prático do uso desse padrão seria:

“Como **um vendedor de livros** eu quero **procurar por livros filtrando por nome** para que seja possível **verificar se um livro X está disponível na loja”**

Cartão	
Como um vendedor de livros eu quero procurar por livros filtrando por nome para que seja possível verificar se um livro X está disponível na loja	
Valor: Médio Risco: Baixo	Estimativa: 3 pontos

Figura 25 – Exemplo de uma User Storie em um cartão

Qualquer pessoa envolvida no projeto pode escrever as *User Stories*, porém, principalmente nas primeiras reuniões, os clientes ou futuros usuários do sistema são os responsáveis por escrevê-las, Cohn (2004) aponta duas razões principais para isso. Primeiro cada estória deve ser escrita na linguagem dos negócios, não na linguagem técnica, para que a equipe do cliente pode priorizar as estórias para inclusão em iterações e versões. Segundo, como os visionares do produto primário, a equipe do cliente está na melhor posição para descrever o comportamento do produto.

De acordo com Cohn (2004), para criar uma boa estória são necessários seis atributos:

- **Independente:** devem ser evitadas dependências entre as estórias, pois isso pode levar a problemas com o planejamento e priorização entre as estórias.
- **Negociável:** estórias não devem fazer parte de um contrato de serviços, dessa forma elas devem ser negociáveis, podendo sofrer alterações.
- **Valiosa para os usuários ou clientes:** algumas estórias podem ser importantes para a equipe de desenvolvimento, como por exemplo, os tipos de conexões de banco de dados que serão utilizadas ou o nível CMM que conduzirá o projeto, mas que não são importantes para os usuários. Esses tipos de estórias devem ser evitadas.
- **Estimável:** as estórias precisam ser estimáveis para que os desenvolvedores possam estipular um tempo de entrega da mesma, caso haja algum impedimento técnico ou de entendimento, isto deve ser esclarecido.
- **Pequena:** as estórias devem ser pequenas no sentido de não haver múltiplas estórias em um único item, pois isso torna difícil o planejamento.
- **Testável:** Estórias devem ser escritas de forma a serem testadas. Uma estória como “Um usuário nunca deve ter de esperar muito para qualquer tela a aparecer” não é testável porque ele diz que "nunca" e não define o que é "esperar muito", nesse caso seria mais razoável escrever “Novas telas devem aparecer dentro de dois segundos, em 95% de todos os casos”.

As *User Stories* foram criadas para serem simples, mas é claro que não é possível realizar o desenvolvimento somente com elas. Faz parte do processo a conversação e a confirmação.

3.5.2 Conversação e Confirmação

A conversação representa uma discussão entre a equipe, clientes, usuários finais e outras partes interessadas, que esclarece os detalhes do requisito estabelecidos nos cartões em forma de *User Stories* e protótipos da solução a ser utilizada.

O termo conversação reflete a natureza da interação verbal - a negociação em torno dos requisitos é através de um diálogo rico e altamente interativo, utilizando

um vocabulário compartilhado compreensível para os clientes e para a equipe de desenvolvimento. A conversação não necessariamente resulta em especificação escrita (ABRAHAMSSON; OZA, 2010), mas algumas observações consideradas importantes podem ser inclusas no cartão em forma de notas simples.

A confirmação representa os critérios de aceitação ou testes, que devem ser satisfeitas antes que a estória possa ser considerada totalmente implementada. Ao contrário da conversa, tais testes são normalmente escritos e documentados para consulta posterior muitas vezes no verso do cartão da *User Story*. Ao assegurar que estes testes passaram, a equipe de desenvolvimento deve estar confiante de que o valor da estória foi entregue ao cliente (ABRAHAMSSON; OZA, 2010).

Dessa forma, enquanto o cartão contém o texto resumido com a estória, os detalhes são trabalhados na conversa e registrados na confirmação.

3.5.3 Prós e Contras da modelagem com User Stories

A especificação de requisitos utilizando *User Stories* reduz consideravelmente o tempo de definição dos requisitos, visto que não é necessária uma documentação formal, extensa e elaborada com as funcionalidades acordadas, já que a maior parte do processo de definição é realizada através de conversas.

Esta técnica auxilia na definição da estimativa de tempo e esforço, pois o problema é fracionado em pequenas partes, facilitando a correta análise dos desenvolvedores.

As *User Stories* enfatizam o uso da comunicação verbal o que possibilita uma melhor comunicação entre as partes interessadas, além de diminuir a ambiguidade que a linguagem escrita contém. Muitas vezes o que é falado fica muito bem entendido entre as partes, mas ao ser formalizado, gera uma série de imprecisões.

Usualmente os próprios usuários ou compradores do sistema almejarem um software, mas não sabem exatamente o que querem, nesse sentido as *User Stories* são úteis, pois inicialmente não é necessário explanar todos os detalhes da aplicação. Além disso, a técnica é flexível no sentido de suportar alterações de escopo com custos menores.

A falta de uma documentação completa e formal pode ser impeditiva para alguns projetos e organizações, caso este item seja considerado imprescindível como um contrato de desenvolvimento para uma das partes.

A técnica exige a participação constante das partes interessadas, o que inclui os clientes ou usuários e compradores, durante todo o projeto. Essa disponibilidade nem sempre é facilmente adquirida, devendo estar devidamente acordada para garantir a qualidade e sucesso do projeto.

User Stories é uma excelente opção para o desenvolvimento iterativo, ou seja, com entregas ou iterações de duas ou no máximo quatro semanas, devido grande parte da análise de requisitos ser baseada em conversas. Dessa forma, é impraticável sua aplicação em desenvolvimentos do estilo cascata, no qual todo o levantamento de requisitos do sistema a ser entregue é realizado de uma vez só.

3.6 Comparação entre as técnicas

Será apresentada nesta seção uma comparação detalhada entre as técnicas estudadas neste trabalho, focando nos seguintes contextos:

- **Requisitos funcionais e não funcionais:** discute se as técnicas possuem suporte para especificar requisitos funcionais e não funcionais, sendo estes os principais requisitos de um software.
- **Ambiguidade:** discute o grau de ambiguidade presente em cada tipo de modelagem, visto que este é um ponto enfatizado nas vantagens e desvantagens de cada um dos modelos.
- **Legibilidade para o usuário:** discute o grau de legibilidade das técnicas para os usuários e clientes do sistema.
- **Facilidade de aprendizagem:** discute o quão fácil os métodos são de aprender levando em conta o ponto de vista técnico.
- **Automatização da validação:** apresenta as técnicas que dão suporte a alguma forma de validação automática, permitindo a diminuição da ambiguidade e inconsistências do modelo.
- **Campo de aplicação:** apresenta os tipos e tamanho de projetos e as metodologias de desenvolvimento que os modelos melhor se adaptam.

A tabela 2 exibe um resumo dos itens que serão discutidos nesta seção (apresentados acima), classificados da seguinte forma:

- 1: significa que a técnica atende totalmente o requisito.
- 2: significa que a técnica atende parcialmente o requisito.
- 3: significa que a técnica não atende o requisito.

Tabela 2 – Comparação entre as técnicas de modelagem de requisitos

	Casos de Usos	SADT	Redes De Petri	SREM	User Stories
Requisitos Funcionais	1	1	1	1	1
Requisitos Não Funcionais	3	1	1	1	2
Não Ambíguo	3	2	1	1	2
Legível para o Usuário	1	2	3	3	1
Facilidade de Aprendizagem	2	2	3	3	1
Automatização da Validação	3	2	1	1	3

As próximas seções detalham e justificam a classificação apresentada na tabela 2.

3.6.1 Requisitos funcionais e Não funcionais

Todas as técnicas apresentadas comportam a especificação dos requisitos funcionais.

O caso de uso é uma técnica utilizada para modelar exclusivamente os requisitos funcionais do sistema, existindo diversos estudos de casos que comprovam a sua eficiência, se corretamente aplicado, para atingir esse objetivo. Porém ela não comporta a descrição de requisitos não funcionais, sendo necessária a utilização de outros documentos complementares para isso.

As *User Stories* descrevem funcionalidades que possuem valor para um usuário do software, portanto podem contemplar alguns requisitos não funcionais, como por exemplo, o desempenho, mas não garantem e não faz parte de seu objetivo que todos os requisitos não funcionais sejam mapeados através da técnica.

As técnicas de modelagem SADT, Redes de Petri e SREM permitem a descrição dos requisitos não funcionais em seus modelos. SADT representa os requisitos não funcionais explicitamente pela seta de controle, que determina de que maneira uma atividade é feita ou restringe algo.

Diversos modelos de redes de Petri podem ser utilizados para modelar os requisitos não funcionais, tais como redes de Petri temporizadas e redes de Petri Colorida.

SREM facilita a especificação formal dos requisitos não funcionais, como os referentes às restrições de tempo de processamento, devido à utilização de estímulo-resposta.

3.6.2 Ambiguidade

Todas as técnicas apresentadas neste trabalho necessitam da intervenção humana para serem aplicadas e desenvolvidas, logo a ambiguidade pode estar presente em todas. Entretanto, algumas dessas apresentam um modelo e validações que diminui o grau de ambiguidade consideravelmente.

Apesar de uma das principais vantagens dos casos de uso ser o fato da utilização da linguagem natural, facilitando a comunicação entre as partes envolvidas, esse vem a ser um de seus principais problemas. A linguagem natural e principalmente, escrita, traz consigo um alto grau de ambiguidade. Mesmo com diversos métodos sugeridos para evitar que alguma imprecisão ocorra nas especificações, muitas vezes, para o autor é tão claro o que ele quer expressar que a confusão das palavras utilizadas não é percebida por ele. Um bom exemplo de ambiguidade é apresentado na seguinte frase (GERHARDT; SILVEIRA, 2009) “Eu vi os anúncios dos tênis Nike, mas não gostei deles”, não fica claro do que a pessoa não gostou, se foram dos anúncios ou se foram dos tênis.

SADT, apesar de utilizar também a linguagem natural, propõe um conjunto prático de ferramentas de modelagem gráfica que superaram as deficiências evidentes da especificação narrativa. Porém, a sua complexidade e nível de detalhamento podem atrapalhar o entendimento do modelo.

User Stories também utilizam a linguagem natural, contudo o seu processo define como a maior parte da definição de requisitos a conversação, ou seja, pela comunicação verbal, o que diminui a ambiguidade que a linguagem escrita contém. Além disso, essa técnica está embutida em metodologias que possuem um clico de validação por interação, que permite a avaliação do cliente durante diversos ciclos do projeto, adiantando qualquer tipo de discordância.

Sobretudo, nenhuma das técnicas é considerada menos ambígua dos que as fundamentadas em métodos formais, como Redes de Petri e SREM. Os métodos

formais levam a uma definição precisa, prevenindo ambiguidades, incertezas e contradições.

3.6.3 Legibilidade para o Usuário

Casos de uso e *User Stories* são facilmente entendidos pelos usuários finais e clientes, devido ao uso da linguagem natural como base.

Apesar do SADT permitir e especificar a utilização da linguagem natural em seus diagramas e anotações, a sua complexidade e nível de detalhamento podem atrapalhar o entendimento do modelo e pode ser difícil para os usuários de apreciá-lo como um modelo adequado.

SREM e Redes de Petri utilizam métodos formais e dificilmente são compreendidas pelos clientes e usuários que tendem a evitar essa técnica por não serem familiarizadas com a mesma e não as financiam.

3.6.4 Facilidade de Aprendizagem

Todas as técnicas necessitam de treinamento para serem utilizadas e aplicadas corretamente. Entretanto, algumas dessas são mais fáceis de aprender e ensinar e não necessitam de conhecimento técnico para isso.

Os casos de uso podem ser facilmente aprendidos e aplicados, mas essa visão leva ao entendimento de que não é necessário nenhum treinamento, acarretando na má modelagem do sistema, que muitas vezes acaba não sendo baseado em funções e nem na interação usuário sistema. Por isso, é necessário um estudo técnico mínimo para modelar com esse método.

O método SADT é apresentado de diversos pontos de vista e níveis de detalhamento, para entender os seus níveis hierárquicos, diagramas e aplicá-los corretamente também é necessário um estudo técnico sobre a linguagem.

Redes de Petri e SREM são métodos formais baseadas em linguagem matemática e de programação, por isso exigem um treinamento específico que demanda maior tempo de aprendizagem.

Já as *User Stories*, entre as técnicas apresentadas, é a mais fácil de entender e aplicar, tanto que elas são escritas grande parte das vezes, pelos próprios usuários e clientes do sistema.

3.6.5 Automatização da Validação

Os casos de uso os e *User Stories* não permitem automatização de suas especificações e modelagens, mas é preciso que fique claro que o objetivo e características dessas técnicas não fazem o menor sentido de serem automatizadas, visto que são baseadas em linguagem natural, sendo ela escrita ou verbal.

Apesar de não ser o seu objetivo principal, é possível automatizar o SADT com o uso do PSA/PSL. PSL estabelece os requisitos para projetar um sistema. O PSA é um analisador automático, o qual é responsável por fornecer os dados que foram previamente obtidos com PSL.

As Redes de Petri, apresentam uma modelo matemático e lógico, provendo uma auto validação. Existem diversas técnicas disponíveis para a validação do processo, entre elas utilizando o *Extensible Markup Language* (XML), que permite a leitura de dados, geração de resultados e interação com outras aplicações (PAIS, 2004).

O SREM utiliza o REVS que é formado por um conjunto de ferramentas que permitem processar o RSL para efetuar a análise do banco de dados e verificar a consistência e integridade dos requisitos, apresentando uma análise automática das incoerências encontradas.

3.6.6 Campo de Aplicação

Os casos de uso são indicados para projetos de softwares em que sejam determinantes as interações entre atores e o sistema a ser desenvolvido. Casos de uso são menos adequados para sistemas que sejam baseadas em funcionalidades que não dependam de uma entrada significante de um ator, como por exemplo, sistemas embarcados.

Os casos de uso podem ser aplicados no método de desenvolvimento ágil ou incremental, onde são escolhidos os casos de usos que serão implementados em cada iteração. Além disso, eles podem ser aplicados no modelo clássico ou cascata, no qual toda a documentação é produzida no inicio do projeto, antes de qualquer desenvolvimento.

O SADT é recomendado para projetos pequenos ou médios, devido a sua complexidade e nível de detalhamento, pois os procedimentos de diagramação podem levar muito tempo e diminuir a flexibilidade de comunicação. Além disso, manter a documentação em dia requer muito tempo e bastante disciplina. Porém

essa não é uma restrição imposta, se estes pontos não forem problemas dentro da estrutura organizada para o projeto, ele pode ser aplicado para grandes softwares.

O SADT é uma análise estruturada, logo é recomendado para projetos que utilizem a metodologia estruturada, pois esta técnica depende fortemente de uma estrutura em camadas hierárquicas para a revelação gradual de detalhes.

As redes de Petri utilizam métodos formais e permitem o desenho de softwares mais complexos de forma precisa e garantem que este cumpre os requisitos pedidos. Para utilizar e entender o modelo, é necessário treinamento e habilidade com a linguagem matemática. Dessa forma, ela não é recomendada para projetos que não exijam esse grau de formalidade.

Muitas vezes esse modelo é utilizado em combinação com outros, como por exemplo, os casos de uso, que utilizam o diagrama de atividades para complementar as suas especificações.

Algumas metodologias de software são difíceis de especificar com as redes de Petri, tais como sistemas interativos e sistemas concorrentes.

SREM normalmente apresenta um melhor custo-benefício se utilizado na especificação de sistemas grandes, integrados e de tempo real. A metodologia é mais adequada para o controle intensivo do sistema (por exemplo, controle de um míssil), em vez de dados intensivos do sistema (por exemplo, processamento de folha de pagamento).

As *User Stories* podem ser aplicadas em projetos de pequeno ou grande porte, mas são recomendadas para o desenvolvimento iterativo e/ou ágil, onde a especificação de requisitos e as entregas são feitas ao longo do projeto de forma contínua, isso porque grande parte da análise de requisitos é baseada em conversas.

4. Considerações finais

Este capítulo apresenta as considerações finais sobre o trabalho e pesquisas futuras ou relacionadas a esta que podem ser realizadas.

4.1 Conclusões

Os requisitos de um software podem ser considerados a parte mais importante em um projeto de desenvolvimento de sistema. Nenhuma outra parte é tão difícil quanto desenhar os requisitos técnicos e funcionais de forma detalhada, incluindo todas as interfaces de modo a atender as necessidades do cliente e ser evidente e clara para a equipe de desenvolvimento, com a finalidade de não haverem implementações incorretas, devido a falta de informações ou a presença dessas de forma ambígua.

Este trabalho teve como objetivo apresentar, analisar e comparar diversas técnicas de modelagem de requisitos com exemplos de suas respectivas documentações, prós e contras. O objetivo deste trabalho foi atingido com a apresentação das técnicas no capítulo 3, demonstrando os suas principais finalidades, aplicações, prós e contras e comparações entre as mesmas.

A pesquisa permitiu identificar os tipos de requisitos que são atendidos por cada técnica com foco nos requisitos funcionais e não-funcionais e o nível de ambiguidade presente em cada uma delas.

Foi apresentado ainda para cada técnica o nível de legibilidade para o usuário, ou seja, o quão fácil a técnica é de ser entendida pelo usuário final ou cliente e o nível de facilidade de aprendizagem, levando em conta o ponto de vista do especialista, isto é, do analista que irá efetuar o desenho da modelagem. Conseguiu-se verificar um padrão interessante nestes pontos. Quanto menos ambígua é a técnica, mais difícil é o seu entendimento pelos usuários e a sua aprendizagem para os técnicos.

Visando a validação dos modelos, foram apresentadas as técnicas que possuem algum suporte quanto a automatização da validação dos requisitos, visando a diminuição das incoerências possíveis nos modelos. Novamente neste item foi encontrado um padrão relacionado à imprecisão, somente as técnicas com

menor grau de ambiguidade e com métodos formais presentes permitem a automação da validação.

Por fim, a pesquisa permitiu identificar o campo de aplicação dos modelos estudados, apresentando os tipos de projetos de software para os quais as técnicas são recomendas e mais eficazes, levando em conta o tamanho do projeto e método de desenvolvimento utilizado.

Não foi a pretensão deste trabalho indicar que uma técnica de modelagem de requisitos é melhor do que as demais e sim proporcionar uma base para o leitor conhecer uma variedade de modelos e definir os formatos mais adequados para seus projetos, visando minimizar a problemática apontada com a modelagem de requisitos. Todos os modelos são potencialmente aplicáveis e possuem suas restrições.

4.2 Extensões Futuras

As extensões que podem ser feitas a este trabalho são as seguintes:

- Realização de pesquisas buscando a solução para os problemas apresentados em cada uma das técnicas.
- Análises de técnicas específicas para cada metodologia de desenvolvimento, como por exemplo, modelo incremental, desenvolvimento ágil ou métodos formais.
- Eleger uma técnica para efetuar a análise detalhada, não apenas a apresentando, mas instruindo a aplicação da mesma, proporcionando um estudo de caso.

5. Referências Bibliográficas

AALST, W. V. D.; HEE, K. V. **Gestão de Workflows:** Modelos, métodos e sistemas. Coimbra: Imprensa da Universidade de Coimbra, 2009. 316p.

ABRAHAMSSON, P.; OZA, N. **Lean Enterprise Software and Systems**. Nova York: Springer, 2010. 185p.

AKTAS, A. Z. **Structured analysis and design of information systems.** Nova Jersey: Prentice-Hall, 1987. 190p.

ALFORD, M. W. **A Requirements Engineering Methodology for Real-Time Processing Requirements**. IEEE Computer, v.SE-3, n.1, p.60-69, 1977. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1702403&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1702403>. Acesso em: 11 dez. 2012.

AMBLER, S. W.; CHWIF, L. **Modelagem Ágil.** Porto Alegre: Artmed, 2002. 351p.

AMO, F. A.; LOIC, A. M. N.; PÉREZ, F. J. S. **Introducción a la Ingeniería del Software:** Modelos de desarollo de programas. Saragoça: Delta, 2005. 283p.

ANDERSON, C.; DORFMAN, M. **Aerospace Software Engineering: A Collection of Concepts**. v. 136. Washington: AIAA, 1991. 631p.

BARROS, J. P. M. P. R. **Introdução à modelação de sistemas utilizando redes de Petri**. Instituto Politécnico de Beja Escola Superior de Tecnologia e Gestão, 2001. Disponível em <http://www.estig.ipbeja.pt/~jpb/textos/pn.pdf>. Acesso em 03 dez. 2012.

BEZERRA, E. **Princípios de Analise e Projeto de Sistemas Com Uml**. Rio de Janeiro: Elsevier, 2002. 374p.

BIRREL, N. D.; OULD, M. A. **A Practical Handbook for Software Development.** Nova York: Cambridge University Press, 1985. 272p.

BLASCHEK. J.R. **Gerência de Requisitos**: O principal problema dos projetos de software. Developers CIO Magazine**,** Rio de Janeiro, v.70, p. 46-47, 2002.

BOEHM, B. W. **Verifying and Validating Software Requirements and Design Specifications**. IEEE Software, p.75-88, 1984. Disponível em: <http://csse.usc.edu/csse/TECHRPTS/1984/usccse84-502/usccse84-502.pdf>. Acesso em: 09 dez. 2012.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML Guia do Usuário.** 2. ed. Rio de Janeiro: Elsevier, 2006. 474p.

BRAUDE, E. **Projeto de Software -** Da programação à arquitetura: uma abordagem baseada em Java. Porto Alegre: Artmed, 2004. 619p.

BROOKS, F. P. **No Silver Bullet**. IEEE Computer, v.20, n.4, p.10-19, 1987. Disponível em: <http://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet>. Acesso em: 20 set. 2012.

CARDOSO, J.; VALETTE, R. **Redes de Petri.** Florianópolis: UFSC, 1997. 212p.

CASTRO, J. **Análise e Modelagem de requisitos**. 2001. Disponível em: <http://www.cin.ufpe.br/~if119/aulas/6-ReqAna.pdf>. Acesso em: 19 nov. 2012.

CHAAR, J. K. **Software Design Methodologies**. Department of Electrical Engineering and Computer Science The University of Michigan, 1987.

CHEN, T. T. **Information management in integrated information system development environments**. The University of Arizona, 1988. Disponível em < http://www.agileinnovation.eu/Library/AI_Publications/User%20Stories%20and%20Innovation%20Spaces%20-%20Final.pdf >. Acesso em 12 dez. 2012.

COHN, M. **User Stories Applied:** For Agile Software Development. Boston: Addison-Wesley, 2004. 268p.

DEZERBELLES, S. P. **A Importância da Análise de Requisitos no Desenvolvimento de Softwares**. Cabo Frio: 2008. Trabalho apresentado à Universidade Veiga de Almeida. Disponível em: <http://pt.scribd.com/doc/16294541/A-importancia-da-analise-de-requisitos-no-desenvolvimento-de-Softwares-by-Swellen>. Acesso em: 19 nov. 2012.

DOUGLASS, B. P. **Real Time UML Third Edition:** Advances in the UML for real-time Systems. 3 ed. Boston: Addison-Wesley, 2004. 694p.

EOCHA,C. O.; CONBOY, K. **The Role of the User Story Agile Practice in Innovation**. National University of Ireland Galway, 2010. Disponível em < http://arizona.openrepository.com/arizona/bitstream/10150/184352/1/azu_td_8814221_sip1_m.pdf>. Acesso em 12 dez. 2012.

FIGUEIREDO, C. et al. **Especificação Formal de Software**. Faculdade de Engenharia da Universidade do Porto, 2002. Disponível em: <http://paginas.fe.up.pt/~ei99030/trabalhos/ApresentacaoES.pdf>. Acesso em: 09 dez. 2012.

FIRESMITH, D. G. **Use Cases: the Pros and Cons**. Wisdom of the Gurus: A Vision for Object Technology, Nova York, p.171-180, 1996. Disponível em: <http://www.ksc.com/article7.htm>. Acesso em: 22 set. 2012.

FORTUNA, M. H.; BORGES, M. R. S. **Modelagem Informacional de Requisitos**. Rio de Janeiro: 2005. Universidade Federal do Rio de Janeiro. Disponível em: <http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/michel_fortuna.pdf>. Acesso em: 24 nov. 2012.

FRANCÊS, C. R. L. **Introdução às Redes de Petri**. Universidade Federal do Pará - UFPA. 2003. Disponível em: <http://www.dca.ufrn.br/~affonso/DCA0409/pdf/redes_de_petri.pdf >. Acesso em: 25 nov. 2012.

FURTADO, V. **Tecnologia e Gestão da Informação na Segurança Pública**. Rio de Janeiro: Garamond, 2002. 261p.

GERHARDT, T. E.; SILVEIRA, D. T. **Métodos de Pesquisa**. 1.ed. Rio Grande do Sul: UFRGS, 2009. 120p.

GUIMARÃES, J.O. **Redes de Petri**. Departamento de Computação Universidade Federal de São Carlos, 2003. Disponível em <http://www2.joinville.udesc.br/~miltonh/mfo/Materiais/jose_Petri_ORIGINAL.pdf>. Acesso em 02 dez. 2012.

IIBA. **Guia BABOK: O Guia para o Corpo de Conhecimento de Análise de Negócios**. Versão 2. Rio de Janeiro: do Autor, 2011. 266p.

ISO. **ISO 8402**. International Standards Organization.1994. Disponível em: <http://pt.scribd.com/doc/40047151/ISO-8402-1994-ISO-Definitions>. Acesso em: 20 nov. 2012.

JACOBSON, I. **Object-Oriented Software Engineering:** a use case driven approach. 1. ed. Boston: Addison-Wesley, 1992. 552p.

KÜNDIG, A.; BÜHRER, R. E.; DÄHLER, J. **Lecture Notes in Computer Science.** Berlim: Springer-Verlag, 1987. 207p.

LARA, W. A. R. **Sistema de Informacion Gerencial para Almacenadoras**. Guatemala: 1992. Dissertação apresentada à Universidade Francisco Marroquin.

LARMAN, G. **Utilizando UML e Padrões.** 3. ed. Porto Alegre: Bookman, 2005. 695p.

LEFFINGWELL, D.; WIDRIG, D. **Managing Software Requirements:** A Unified Approach**.** 1. ed. Boston: Addison-Wesley, 1999. 528p.

MARTINS, J. C. C. **Técnicas para gerenciamento de projetos de software.** Rio de Janeiro: Brasport, 2007. 456p.

MEDINA, A. C.; CHWIF, L. **Modelagem e Simulação de Eventos Discretos:** Teoria e Aplicações**.** 3. ed. São Paulo: do Autor, 2010. 320p.

MELO, M. **Guia de Estudo para o exame PMP:** Project Management Professional Exam. 4. ed. Rio de Janeiro: Brasport, 2012. 608p.

MONTEZANO, A. F. M. **Modelo em Rede de Petri de um Sistema de Automação de Elevador de Passageiros**. Rio de Janeiro: 2009. Dissertação apresentada à Escola Politécnica da Universidade Federal do Rio de Janeiro.

MOURA, A. V. **Especificações em Z, uma introdução**, 1. Ed. Campinas: UNICAMP, 2001. 306p.

NETO, A. L. B.; SANTOS, R. F. **Métodos e Técnicas de Desenvolvimento**. São Carlos: 2012. Trabalho apresentado à Universidade Federal de São Carlos. Disponível em: <www2.dc.ufscar.br/~rosangela/ES/Tecnicas.pot>. Acesso em: 26 nov. 2012.

OAKLAND, J. S. **Gerenciamento da Qualidade Total**. São Paulo: Nobel, 1994. 459p.

OUCHI, W. **Teoria Z:** Como as empresas podem enfrentar o desafio japonês. 10. ed. São Paulo: Nobel, 1986. 293p.

PÁDUA, S. I. D. et al. **O potencial das redes de Petri em modelagem e análise de processos de negócio**. Universidade Federal de São Carlos, v.11, n.1, 2004. Disponível em: <http://www.scielo.br/scielo.php?pid=S0104-530X2004000100010&script=sci_arttext >. Acesso em: 01 dez. 2012.

PAIS, R. M. C. **Geração de Executores e Analisadores de Redes de Petri.** Universidade Nova de Lisboa – Departamento de Informática, 2004. Disponível em <http://www.estig.ipbeja.pt/~rmcp/pdf/TeseMestrado_RuiPais.pdf>. Acesso em 13 dez. 2012.

PALOMINO, R. C. **Uma Abordagem para a Modelagem, Análise e Controle de Sistemas de Produção Utilizando Redes de Petri**. Florianópolis: 1995. Dissertação apresentada à Universidade Federal de Santa Catarina.

PREECE, J.; ROGERS, Y.; SHARP, H. **Design de Interação:** Além da interação homem-computador**.** Porto Alegre: Bookman, 2002. 548p.

QIDIGITAL. **Especificação do Caso de Uso Controlar Figura.** Portal QiDigital. 2011. Disponível em: <http://pt.scribd.com/doc/56208660/1%C2%BA-Exemplo-de-Especificacao-de-Caso-de-Uso>. Acesso em: 23 nov. 2012.

RAMOS, R. A. **Treinamento Prático em UML.** São Paulo: Digerati Books, 2006. 144p.

RAMPAZZO, L. **Metodologia Científica.** 3. ed. São Paulo: Loyola, 2005. 141p.

REZENDE, D. A. **Engenharia de Software e Sistema de Informação.** 3. ed. Rio de Janeiro: Brasport, 2005. 316p.

RIBEIRO, A. L. **Semântica e uma ferramenta para o método SADT**. Porto Alegre: 1991. Dissertação apresentada à Universidade Federal do Rio Grande do Sul.

SANTANA, M. J.; SANTANA, R. H. C. **Modelagem: Técnicas de Avaliação de Desempenho**. Universidade de São Paulo - Departamento de Sistemas de Computação, 2011. Disponível em: <http://wiki.icmc.usp.br/images/1/13/Aula4_tecnicas_modelagem.pdf>. Acesso em: 09 dez. 2012.

SAYÃO, M.; STAA, A. V.; LEITE, J. C. S. P. **Qualidade em Requisitos.** Rio de Janeiro: PUC-Rio Departamento de Informática, 2003.

STANDISH. **Chaos Report 1995**. Standish Group. 1995. Disponível em: <http://www.projectsmart.co.uk/docs/chaos-report.pdf>. Acesso em: 20 set. 2012.

TANENBAUM, A. S. **Redes de Computadores**, 4. Ed. Rio de Janeiro: Elsevier, 2003. 945p.

TOVAR, E. **Modelação e Análise de Sistemas Computacionais**. Departamento de Engenharia Informática Instituto Superior de Engenharia do Porto, 2007. Disponível em <http://www.dei.isep.ipp.pt/~npereira/aulas/compa/09/COMPA-MOD1-TEORICA_v1.pdf>. Acesso em 08 dez. 2012.

TSE, T. H.; PONG, L. **An Examination of Requirements Specification Languages**. Department of Computer Science The University of Hong Kong. 1989. Disponível em: <http://www.csis.hku.hk/research/techreps/document/TR-89-09.pdf>. Acesso em: 11 dez. 2012.

WIKISPACES. **Máquinas de Estado Finito (MEF)**. Wikispaces, 2009. Disponível em < http://sed-ferias09-20610043.wikispaces.com/02.+M%C3%A1quinas+de+Estado+Finito+(MEF)>. Acesso em 07 dez. 2012.