Pontifícia Universidade Católica de São Paulo

Fernanda Lins Reis

Análise da Engenharia de Sistemas no Desenvolvimento de Softwares de Automação

São Paulo

2009

Pontifícia Universidade Católica de São Paulo

Fernanda Lins Reis

Análise da Engenharia de Sistemas no Desenvolvimento de Softwares de Automação

Monografia apresentada para conclusão
do curso de Especialização em
Engenharia de Software.

Orientador: Prof. Dr. Carlos Eduardo de Barros Paes.

São Paulo

2009

Fernanda Lins Reis

Análise da Engenharia de Sistemas no Desenvolvimento de Softwares de Automação

Monografia apresentada para conclusão
do curso de Especialização em
Engenharia de Software.

Orientador: Prof. Dr. Carlos Eduardo de Barros Paes.

São Paulo, ...de.....de 2009

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

Albert Einstein

RESUMO

Este trabalho tem por objetivo realizar um estudo sobre a Engenharia de Sistemas e apresentar seus princípios básicos, assim como as soluções existentes para atender as necessidades das metodologias de desenvolvimento, das linguagens de modelagem e dos modelos de maturidade.

A Engenharia de Sistemas é uma importante área que permite aplicar os princípios da Engenharia no desenvolvimento de sistemas complexos. Outra importante característica da Engenharia de Sistemas é preocupar-se não somente com o software, mas também com o hardware, com o ambiente o qual o sistema pertencerá e com os usuários envolvidos. Desta forma, possui um escopo mais amplo que a Engenharia de Software.

Diante destas características, é possível observar que os sistemas de automação podem ser considerados bons exemplos de sistemas complexos, cujo desenvolvimento envolve não somente o software, mas outros elementos fundamentais. Desta forma, podemos considerar a Engenharia de Sistemas durante o processo de desenvolvimento de sistemas de automação.

Também faz parte do escopo deste trabalho, o entendimento dos conceitos relacionados à Engenharia de Sistema, através de um comparativo dos conceitos relacionados à Engenharia de Software.

ABSTRACT

This study aims to conduct a search about the Systems Engineering and present its basic principles, as well as the existing solutions to meet the needs of development methodologies, modeling languages and the maturity model.

The Systems Engineering is an important area that allows applying the principles of engineering in the development of complex systems. Another important feature of systems engineering is concerned not only with the software, but also with the hardware, the environment which the system is owned and users involved. Thus, it has a broader scope that the Software Engineering.

Because of these characteristics, you can see that the Automation Systems can be considered good examples of complex systems, whose development involves not only software, but other elements. Thus, we can consider the systems engineering during the development process of the automation systems.

Also is part of the scope of this study, the understanding of concepts related to Engineering System, through a comparison of concepts related to Software Engineering.

ABREVIATURAS

CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
G2SEBoK	Guide to the Systems Engineering Body of Knowledge
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
OMG	Object Management Group
RUP	Rational Unified Process
RUP-SE	Rational Unified Process for Systems Engineering
SDCD	Sistema Digital de Controle Distribuído
SECAM	Systems Engineering Capability Assessment Model
SE-CMM	Systems Engineering Capability Maturity Model
SEI	Software Engineering Institute
SWEBOK	Guide to the Software Engineering Body of Knowlegde
SysML	Systems Modeling Language
UML	Unified Modeling Language

SUMÁRIO

FIGURAS

1 INTRODUÇÃO

Atualmente os softwares de computadores apresentam complexidade cada vez mais alta devido às necessidades de negócio. Os sistemas de automação estão entre os softwares de computadores que exigem desafios para os engenheiros de software, pois além da complexidade dos softwares, os sistemas de automação envolvem conhecimentos em infra-estrutura de comunicação, em diversos níveis e tipos de redes, em sistemas distribuídos de tempo real, entre outros.

De acordo com Nise (2002), os sistemas controlados automaticamente são partes integrantes da sociedade moderna. Encontramos sistemas de controle por toda parte ao longo da indústria de controle de processos regulando níveis de líquidos em reservatórios, concentrações químicas em tonéis, bem como espessura de materiais fabricados. Os sistemas de controle também encontram aplicação ampla na direção, navegação e controle de mísseis e de naves espaciais, bem como em aviões e navios. No setor petroquímico, em muitos processos de operação de refinarias sistemas de automação são necessários.

Segundo Pressman (2006), sistemas baseados em computador interagem com o mundo real de um modo reativo. Isto significa que, eventos do mundo real são monitorados pelo hardware e pelo software que formam o sistema baseado em computador e, com base nesses eventos, o sistema impõe controle sobre as

máquinas, os processos e até as pessoas que causam a ocorrência dos eventos. Muitos sistemas na categoria reativa controlam máquinas e/ou processos (por exemplo, aeronaves comerciais ou refinarias de petróleo) que precisam operar com um grau de confiabilidade extremamente alto. Se o sistema falhar, perdas humanas ou econômicas podem ocorrer.

1.1 MOTIVAÇÃO

No contexto de uma empresa petrolífera, diversas áreas visam obter melhorias em seus processos tendo como objetivo ganhos de resultados de produção e segurança nas operações. A automação destes processos visa dotar aos que atuam diretamente no processo produtivo, meios necessários para tomada de decisões quanto à execução de atividades operacionais que impactam diretamente o negócio.

A área de aplicações em automação das indústrias de processamento químico/petroquímico caracteriza-se pela utilização, de softwares matematicamente complexos com o objetivo de maximizar a lucratividade de instalações industriais, pela definição precisa do seu ponto ótimo de operação e ajuste contínuo das variáveis do processo para levá-lo e mantê-lo neste ponto. Benefícios adicionais são auferidos em função da qualidade mais estável dos produtos, maior estabilidade do processo frente a perturbações e menor demanda de mão de obra de operação.

Devido à alta complexidade dos sistemas de automação, o uso de computadores digitais vem sendo cada vez mais necessário nos desenvolvimentos modernos. E, em empresas cujo negócio exige grande quantidade de sistemas controlados automaticamente, ocorre à necessidade de desenvolver sistemas que em sua maioria são complexos, envolve diversas áreas da empresa, conhecimento em diferentes tecnologias e hardwares, realização de treinamentos e simulações do sistema.

Por exemplo, a área de Transferência e Estocagem de uma refinaria possui diversos sistemas de controle para garantir monitoramento em tempo real de nível e temperatura, automação de plataformas de carregamento, sistemas de mistura em linha, drenagem automática de tanques, dentre outros. Tais sistemas se fazem necessários principalmente para garantir segurança operacional, proteção ambiental, ergonomia, qualidade dos produtos e otimização dos recursos. A Figura 1 ilustra um operador controlando automaticamente a abertura e o fechamento de válvulas.

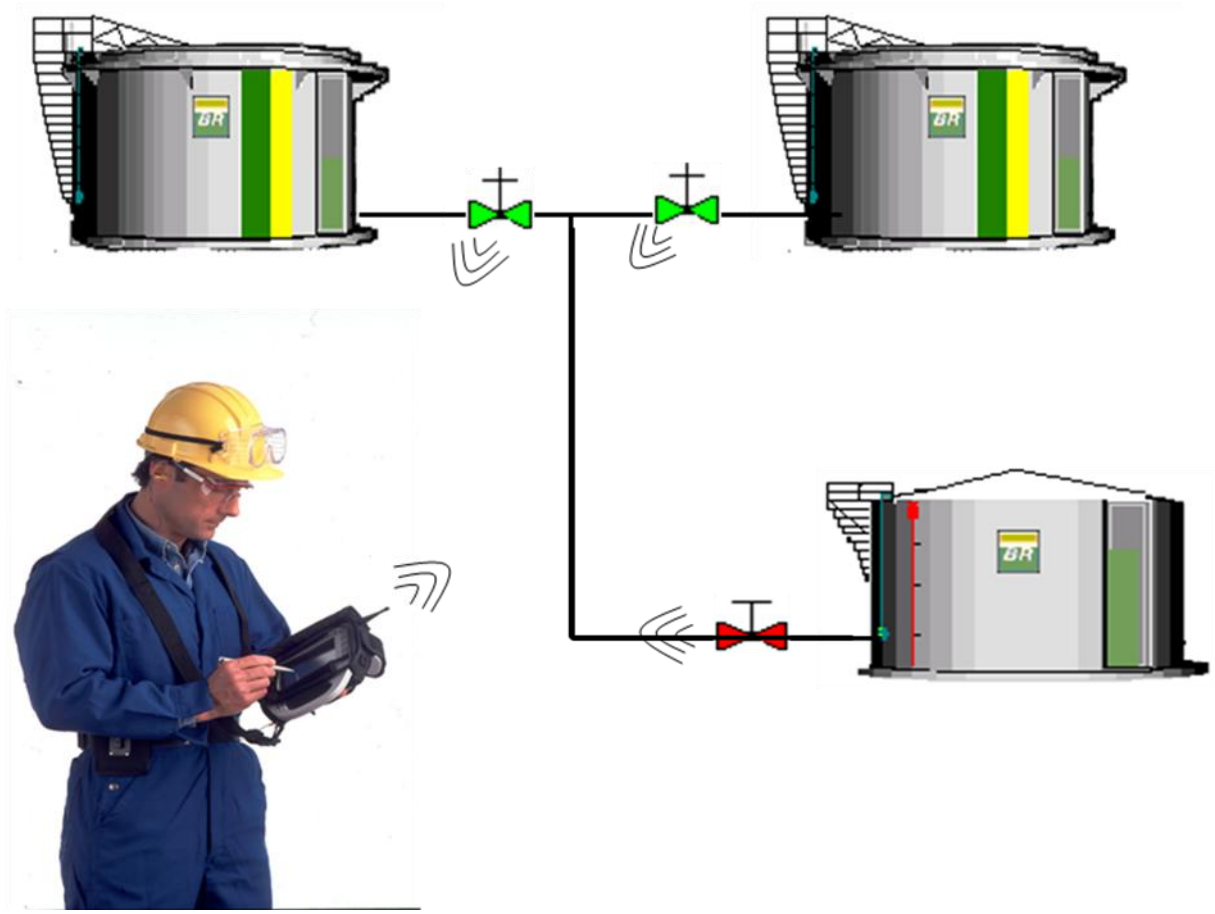

Figura 1 - Representação de equipamentos acionados automaticamente

Na Figura 1, a transmissão dos dados é realizada através de redes *wireless* e *hardwares* que permitem ao operador realizar as seguintes operações:

- Entrar com os dados do processo e as condições dos equipamentos que estão sendo operados (Ex: válvulas de controle);
- Obter informações críticas em tempo real;
- Facilitar e tornar mais seguras atividades como: inspeção de equipamento, tomada de amostras, calibragem e alinhamentos de tanques, entre outras.

A Figura 2 apresenta alguns exemplos de hardwares utilizados para capturar e enviar sinais para a rede *wireless*: Transponders e Tablet PC.

Figura 2 - Tablet PC

Por trabalhar em uma empresa petrolífera, na área de tecnologia da informação e telecomunicações, que desenvolve soluções para as áreas de negócio, atualmente estou alocada em um projeto de desenvolvimento de um sistema para automação dos processos de movimentação e mistura de produtos e que deverá ser implantado em todas as refinarias, com o objetivo de criar um sistema corporativo que atenda a todos os órgãos da empresa.

O desenvolvimento deste sistema envolve processos de diferentes áreas (para integrar os sistemas já existentes), conhecimentos em automação, hardwares que suportem a solução, infra-estrutura de rede e outras tecnologias.

Diante deste cenário, nota-se a necessidade da aplicação de um processo de desenvolvimento que possa se adequar às mudanças de regras de negócio e adaptação de novas tecnologias.

É possível atender demandas deste tipo aplicando a Engenharia de Software? Neste contexto, a Engenharia de Sistemas é mais abrangente e indicada para sistemas com esta complexidade, pois se preocupa com o desenvolvimento de sistemas em grande escala compostos por software, hardware, trabalhadores e componentes de informações, tendo como objetivo o entendimento completo do sistema e o melhor atendimento da necessidade do cliente.

1.2 OBJETIVOS

Este trabalho concentra-se na análise dos processos de desenvolvimento de software que suportem aplicações cujo escopo é abrangente e envolvem diversos elementos além do software em si, procurando consolidar a necessidade da base de conhecimento da engenharia de sistemas. Serão apresentadas as soluções existentes e que suportam as particularidades da engenharia de sistemas, procurando diferenciá-la da engenharia de software.

1.3 ORGANIZAÇÃO DO TRABALHO

No capítulo seguinte, **“Fundamentos e Estado da Arte”,** os principais fundamentos e princípios necessários para o entendimento deste trabalho serão apresentados. Conceitos como: Automação de Sistemas, Engenharia de Sistemas, *UML*, *RUP* e *CMM* serão tratados neste capítulo.

No capítulo 3, “**Soluções para a Engenharia de Sistemas**”, será realizado um estudo sobre a Engenharia de Sistemas e apresentado os processos e ferramentas que suportam esse tipo de desenvolvimento. Será apresentado o ***Rational Unified Process for Systems Engineering (RUP-SE)*** que é uma extensão do *RUP* para a Engenharia de Sistemas. Assim como, será apresentada a extensão da *UML* para a Engenharia de Sistemas (**SysML**). Após a implantação de um processo de engenharia de sistemas é necessário medir a qualidade do processo implantado, para isso existem ferramentas como o ***System Engineering Capability Maturity Modelsm (SE-CMMsm)*** que podem ser utilizadas de maneira variada dependendo do objetivo da companhia. Esta ferramenta também será estudada neste capítulo.

As conclusões deste trabalho estarão presentes no capítulo 4.

Através destes capítulos serão discutidos e apresentados os aspectos que fazem com que a engenharia de sistemas diferencie-se da engenharia de software e o porquê que os processos de desenvolvimento de sistemas de automação devem considerar a engenharia de sistemas.

2 FUNDAMENTOS E ESTADO DA ARTE

O objetivo deste capítulo é apresentar os conceitos relacionados a este trabalho para proporcionar um melhor entendimento do assunto abordado. O capítulo está organizado da seguinte forma: na seção 2.1 apresenta-se uma explicação sobre “Automação de Sistemas”, na seção 2.2 incluem-se as definições sobre “Engenharia de Sistemas”, a seção 2.3 contempla a “Unified Modeling Language”, linguagem de modelagem padrão utilizada no desenvolvimento de softwares e sistemas. A seção 2.4 finaliza o capítulo apresentando o “Rational Unified Process”.

2.1 AUTOMAÇÃO DE SISTEMAS

Automação pode ser entendida como um sistema automático de controle, onde os mecanismos verificam seu próprio funcionamento, realizando medições e correções sem a necessidade da interferência do homem.

Com a automação, técnicas computadorizadas ou mecânicas são aplicadas sobre um determinado processo com o objetivo de torná-lo mais eficiente. Automação visa diminuir custos e aumentar a velocidade da produção.

Automação pode ser aplicada em diferentes áreas. A indústria, por exemplo, aplica técnicas para controle e otimização em seus processos industriais. Na área comercial, utilizam-se mais *softwares* do que *hardwares* e desta forma a automação está presente nos sistemas de controle de estoques, folhas de pagamento, identificação de mercadorias por códigos de barras, etc. Técnicas de automação também estão presentes em residências, através de portões eletrônicos, circuitos fechados de televisão, controle de luminosidade de ambientes, controle de umidade, temperatura e ar condicionado, entre outros.

Segundo Nise (2002), um sistema de controle consiste em subsistemas e processos (ou plantas) reunidos com o propósito de controlar as saídas dos processos. Na sua forma mais simples, um sistema de controle fornece uma saída ou resposta para uma dada entrada ou estímulo, sendo um dos principais benefícios de sua aplicação a possibilidade de mover grandes equipamentos com precisão que, de outra forma seria impossível.

De acordo com Nise (2002), com a evolução da estrutura de automação, o uso de computadores digitais vem sendo comum em determinados sistemas. A função de controlador é exercida por um computador digital permitindo controlar muitas malhas pelo mesmo computador através de forma compartilhada. Além disso, qualquer ajuste requerido nos parâmetros do controlador para produzir uma resposta desejada pode ser feito através de mudanças no software e não no hardware.

Os primeiros sistemas *SDCD* (Sistema Digital de Controle Distribuído) surgiram na década de 70 frutos da necessidade de se ter um controle de processos de forma a permitir uma otimização da produtividade industrial, estruturada na

diminuição de custos de produção, melhoria na qualidade dos produtos, precisão das operações, segurança operacional, entre outros. É através desses sistemas que é possível, por exemplo, obter informações (estados) sobre os equipamentos (bombas, válvulas, etc) presentes em uma refinaria.

Os *SDCDs* consistem de um conjunto de dispositivos microprocessados dedicados ao Controle de Processos, ligados em rede, de forma distribuída. Dotado de processadores e redes redundantes, o *SDCD* controla e supervisiona o processo produtivo. É através das Unidades de Processamento, distribuídas nas áreas, que os sinais dos equipamentos de campo são processados de acordo com a estratégia programada. Estes sinais, transformados em informação de processo, são atualizados em tempo real nas telas de operação das Salas de Controle.

O *SDCD* não é uma máquina pronta. Desta forma, pode ser modificado para atender ao processo. Algumas funções do *SDCD*:

- Indicação/registro de variáveis de processo Contínuas (analógicas) e Discretas (Digitais);
- Capacidade para armazenar históricos em memórias não voláteis;
- Alarmes para variáveis contínuas/discretas, desvio e velocidade de desvio;
- Relatórios de ocorrências de alarmes e eventos;
- Executar funções de Intertravamento relacionados com a segurança dos equipamentos;
- Controle manual das variáveis contínuas e discretas;

- Controle automático tipo PID (proporcional, integral, derivativo) e suas combinações;
- Transferência entre modos Manual/Automático/Cascata/Computador;
- Funções matemáticas diversas;
- Condicionamento dos sinais elétricos e conversão em unidades de engenharia;
- Capacidade para implementar estratégias avançadas de controle e controles lógicos seqüenciais;
- Capacidade de visualização de telas gráficas interativas com indicações dinâmicas;
- Capacidade de interligação com computadores remotos (externos).

Por sua complexidade, e por fazer parte de uma hierarquia de sistemas, ou seja, por ser um sistema baseado em computador que faz parte de um sistema baseado em computador ainda maior, o desenvolvimento de sistemas de automação deve considerar alguns elementos fundamentais que possibilitem atender uma necessidade de negócio ou desenvolver um produto que possa ser utilizado para gerar receita. Considerando a área de Transferência e Estocagem, no contexto de uma refinaria, podemos relacionar os seguintes elementos:

- Software: programas de computador compostos de uma sequência de instruções que são interpretas e executadas por um processador ou por uma máquina virtual. Trata-se da parte lógica do computador. Ex: Softwares para calcular a qualidade de produtos, controlar estoques de produtos,

realizar movimentações de produtos entre tanques, sistemas de mistura em linha, entre outros.

- Hardware: parte física do computador, ou seja, representa o conjunto de componentes eletrônicos, circuitos integrados e placas, que se comunicam através de barramentos. Ex: PDA`s, Tablet PC, coletor de dados.
- Pessoal: usuários e operadores de hardware e de software. Pessoas que interagirão com o sistema. Ex: Operadores da área de transferência e estocagem.
- Banco de Dados: coleção de informações, organizada de forma que se tenha acesso através de software e persista ao longo do tempo.
- Documentação: informações descritivas que detalham o uso e/ou operação do sistema.
- Procedimentos: passos que definem o uso específico de cada elemento do sistema ou o contexto de procedimento no qual o sistema reside.

2.2 ENGENHARIA DE SISTEMAS

A engenharia de sistemas é uma área mais ampla que a engenharia de software, pois cuida de todos os aspectos de sistemas baseados em computadores. Além do software, a engenharia de sistemas preocupa-se com o hardware e com as interações do sistema com usuários e seu ambiente.

Para o INCOSE (*International Council on Systems Engineering*) a Engenharia de Sistemas trata-se de uma abordagem interdisciplinar e significa permitir a realização bem sucedida dos sistemas. Foca na definição das necessidades dos clientes e das funcionalidades necessárias no início do ciclo de desenvolvimento, documentando os requisitos, procedendo com a síntese do projeto e com a validação do sistema, enquanto considera o problema completo.

Segundo Sommerville (2007), existe uma categoria de sistemas classificados como *Sistemas Sociotécnicos*. Tais sistemas possuem processos operacionais definidos, pessoas como partes inerentes do sistema, são regidos por políticas e regras organizacionais e podem ser afetados por restrições externas. A engenharia de sistemas seria a atividade de especificar, projetar, implementar, validar, implantar e manter sistemas deste tipo.

O papel da ciência é determinar “O que é”, através do desenvolvimento de hipóteses teóricas e da experimentação. Toda a engenharia preocupa-se com a aplicação do conhecimento e dos conceitos da ciência e da matemática, buscando criar soluções ou pensar no “Como pode ser”. A engenharia de sistemas trabalha para determinar “Como deveria ser”, ou seja, buscar dentro de um conjunto de requisitos a solução para uma necessidade.

2.2.1 Os Seis Princípios Da Engenharia De Sistemas

Para a *IBM Rational* existem seis princípios da engenharia de sistemas que podem ser considerados como diretrizes obtidas através de resultados da análise do desenvolvimento de sistemas complexos nos últimos anos. Estas diretrizes podem ser utilizadas pelas organizações que desejam obter experiência no desenvolvimento de sistemas que possuem um nível de complexidade elevada. Para a obtenção dessas diretrizes foi observado que o gerenciamento de sistemas complexos deveria ser baseado em três aspectos, a Figura 3 ilustra estes aspectos.

1. Arquitetura do Sistema;
2. Estrutura Organizacional, incluindo a infra-estrutura do desenvolvimento de sistemas;
3. Processos, incluindo fluxos de trabalho, melhores práticas, e outros.

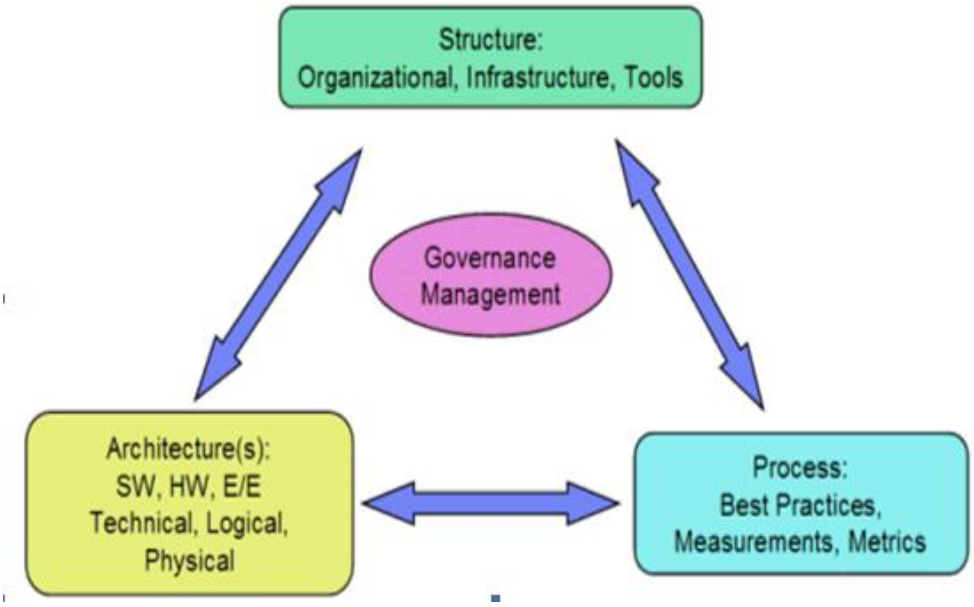

Figura 3 - Aspectos interdependentes para gerenciar o desenvolvimento de sistemas complexos (The Rational Edge)

Os seis princípios propostos pela *IBM Rational* consideram todos os três aspectos, sendo que três destes princípios são técnicos e focam a arquitetura e os modelos de derivação de sistemas. Os demais provêem à infra-estrutura complementar e fluxos de trabalhos necessários para obter o melhor ambiente de desenvolvimento técnico.

Os seis princípios de engenharia de sistemas citados acima são:

1. **Decompor sistemas, não requisitos:** a validação, a atribuição e a análise de requisitos são elementos críticos nesta abordagem, mas não é o foco da análise do sistema, pois tendo como foco principal requisitos, decisões precoces sobre tecnologia e arquitetura terão que ser tomadas. Recomenda-se então, mudar o foco para uma implementação em nível de componentes. O principal objetivo é decompor o sistema estruturalmente em um conjunto de passos lógicos considerando o nível de contexto, de análise, de projeto e de implementação.

2. **Ativar tanto separação e integração das preocupações dos 'principais sistemas de desenvolvimento':** um caminho natural para lidar com a complexidade de sistemas é particioná-lo em unidades menores e menos complexas, no entanto particioná-lo a partir de apenas uma perspectiva não facilitará as atribuições das necessidades de todos os interessados. Engenharia de sistemas não é focada na construção de um sistema que satisfaça necessidades funcionais, mas sim que trate de forma holística o sistema no seu ambiente.

3. **Especificações de fluxos de cima para baixo da arquitetura:** Nem sempre é possível ter um ambiente de desenvolvimento perfeito, onde os processos no nível de contexto, de análise, de projeto e de implementação ocorram de um modo *top-down.* Diversos fatores podem inibir esse fluxo *top-down* da definição do contexto do sistema até sua implementação. Na realidade, as especificações de arquitetura devem cobrir um fluxo de cima para baixo e através das demais visões (elementos de sistemas que fazem parte de outros sistemas), tentando abranger todas as perspectivas do sistema, minimizando que futuras descobertas exijam modificações significativas na arquitetura do sistema.

4. **Sistemas e componentes colaboram, então equipes de desenvolvimento deveriam colaborar:** baseado nos três princípios acima (princípios técnicos), é possível notar a necessidade de alinhamento entre diferentes perspectivas do sistema. Da mesma forma, quando pensamos em equipes de desenvolvimento, os esforços dos profissionais (cada qual em sua área de atuação) também devem ser geridos de maneira a facilitar o desenvolvimento do sistema. Metodologias e ferramentas podem ser utilizadas para auxiliar o gerenciamento dos requisitos, a elaboração dos artefatos e a comunicação entre as equipes. A colaboração entre os times de desenvolvimento pode garantir a interação entre os elementos do sistema provenientes de diferentes pontos de vista.

5. **Organizações de desenvolvimento deveriam refletir a arquitetura dos produtos:** este princípio está relacionado à melhor prática para aumentar a colaboração e a comunicação. Organizações de desenvolvimento

alinhadas a arquitetura dos produtos são eficientes e eficazes. Pois quando existe esse alinhamento natural entre equipes de desenvolvimento, compostos por diversos profissionais (engenheiros, analistas, programadores, testadores, e outros), é possível fornecer competências e formar um banco de recursos. Isso permite que as organizações consigam reagir e re-alinhar quando ocorrem evoluções que envolvem importantes mudanças no conteúdo e na função do produto.

6. **A base no ciclo de vida do desenvolvimento está em remover riscos e adicionar valor:** este último princípio foca em governança, cujo objetivo é garantir que os riscos mais elevados sejam eliminados em primeiro lugar e que o valor do produto/programa que está sendo desenvolvido seja medido regularmente à medida que o desenvolvimento vai progredindo.

2.2.2 Engenharia de Software x Engenharia de Sistemas

Ao se estudar à Engenharia de Sistemas, automaticamente estamos estudando os conceitos relacionados à Engenharia de Software, pois a Engenharia de Sistemas engloba grande parte dos processos realizados ao se desenvolver um software. Podemos considerar que a Engenharia de Sistemas possui um escopo maior que o da Engenharia de Software por considerar outros elementos durante o desenvolvimento do sistema.

Diversas definições tentam explicar o que é a engenharia de software. O termo *Engenharia de Software* passou a ser oficialmente utilizado em 1968 na *NATO Conference on Software Engineering* (Conferência sobre Engenharia de Software da OTAN). Nesta conferência, o cientista da computação Friedrich Ludwig Bauer, definiu que “Engenharia de Software é a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais”.

Segundo Sommerville (2007: 5), “a engenharia de software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em operação”.

A principal diferença entre estas duas áreas está nos elementos envolvidos. A Engenharia de Sistemas preocupa-se em como construir um sistema, que pode conter hardwares, diversos softwares, trabalhadores humanos, diferentes ambientes, fornecendo os recursos operacionais necessários para seus usuários. A Engenharia de Sistemas deve considerar não apenas a especificação e a criação de um produto, mas também o suporte em serviço desse produto (por exemplo, sua manutenção) para assegurar a continuidade de sua utilidade.

Um processo de Engenharia de Sistemas deve, portanto, ser capaz de lidar com sistemas que são complexos e grandes, requerendo uma abordagem multidisciplinar em sua construção, implementação e suporte.

De acordo com Sommerville (2007), a Engenharia de Sistemas preocupa-se com todos os aspectos do desenvolvimento e da evolução de sistemas complexos,

onde o software desempenha um papel importante. Desta forma, além de ser relacionada ao desenvolvimento de hardware, às políticas e aos processos, a Engenharia de Sistemas está diretamente relacionada com a Engenharia de Software.

Outro ponto a ser observado entre estas duas áreas, é que o número de softwares aumentou muito nos sistemas. Devido a este aumento, muitas técnicas de desenvolvimento de softwares surgiram e também são utilizadas no processo de Engenharia de Sistemas. As principais diferenças entre o processo de engenharia de sistemas e o de desenvolvimento de software são apresentadas no Capítulo 3.

2.3 UNIFIED MODELING LANGUAGE (UML)

Durante a construção de um software, modelos podem ser utilizados para identificar as características e as funcionalidades que um software deverá prover. A modelagem de software é uma importante ferramenta que permite construir modelos gráficos para representar os artefatos dos componentes de software utilizados e os seus inter-relacionamentos.

Para que estes modelos possam ser interpretados, uma linguagem de modelagem padronizada é necessária, de forma a fornecer um vocabulário único e permitir a compreensão de um sistema.

A *UML* é uma Linguagem Unificada de Modelagem, ou seja, é uma linguagem gráfica que permite visualizar, especificar, construir e documentar artefatos de

sistemas complexos de software. Por ser independente de processo, é possível utilizá-la com vários processos de engenharia de software.

De acordo com Booch, Rumbaugh e Jacobson (2005: 13), “A UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir sistemas de informação corporativos a serem distribuídos a aplicações baseadas em *Web* e até sistemas complexos embutidos de tempo real”.

Para compreender um modelo conceitual da *UML* é necessário conhecer o vocabulário utilizado nesta linguagem. Este vocabulário abrange três tipos de blocos de construção:

1. Itens: os itens são abstrações identificadas em um modelo. Estas abstrações podem contemplar:

    - Itens estruturais: representam as partes estáticas do modelo, podendo representar elementos conceituais ou físicos. Exemplos: Classes, Interfaces, Colaborações, Casos de uso, Componentes, Nós.

    - Itens comportamentais: representam as partes dinâmicas dos modelos, detalhando os comportamentos no tempo e no espaço. Exemplos: Mensagens, Estados, Ações.

    - Itens de agrupamento: referem-se às partes organizacionais dos modelos. Exemplo: Pacotes.

    - Itens anotacionais: representam as partes explicativas dos modelos. Exemplo: Notas.

2. Relacionamentos: os relacionamentos são os blocos relacionais básicos para a construção dos modelos.

3. Diagramas: é a representação gráfica de um conjunto de elementos. O principal objetivo de um diagrama é permitir a visualização do sistema sobre determinada perspectiva.

A *OMG's Unified Modeling Language™ (UML®)* define com a *UML 2.0,* treze tipos de diagramas divididos em três categorias (estruturais, comportamentais e de interação). A Tabela 1 exibe a descrição dos tipos de diagramas.

Diagrama de Classes	Um diagrama estrutural que mostra um conjunto de classes, interfaces, colaborações e seus relacionamentos.
Diagrama de Objetos	Um diagrama estrutural que mostra um conjunto de objetos e seus relacionamentos.
Diagrama de Componentes	Um diagrama estrutural que mostra as interfaces externas, incluindo portas e a composição interna de um componente.
Diagrama de estrutura composta	Um diagrama estrutural que mostra as interfaces externas e a composição interna de uma classe estruturada.
Diagrama de casos de uso	Um diagrama comportamental que mostra um conjunto de casos de uso e atores e seus relacionamentos.
Diagrama de seqüências	Um diagrama comportamental que mostra uma interação, dando ênfase à ordenação temporal das mensagens.
Diagrama de comunicação	Um diagrama comportamental que mostra uma interação, dando ênfase à organização estrutural de objetos que enviam e recebem mensagens.
Diagrama de gráfico de estados	Um diagrama comportamental que mostra uma máquina de estados, dando ênfase ao comportamento ordenado por eventos de um objeto.
Diagrama de atividades	Um diagrama comportamental que mostra um processo computacional, dando ênfase ao fluxo de uma atividade para outra.
Diagrama de implantação	Um diagrama estrutural que mostra os

	relacionamentos entre um conjunto de nós, artefatos e classes manifestadas e componentes.
Diagrama de pacotes	Um diagrama estrutural que mostra a organização do modelo em pacotes.
Diagrama de temporização	Um diagrama comportamental que mostra uma interação com mensagens em momentos específicos.
Diagrama de visão geral da interação	Um diagrama comportamental que combina aspectos dos diagramas de atividades e dos diagramas de seqüências.

Tabela 1 - Diagramas da UML (Booch et al. 2005).

Apesar da *UML* definir uma linguagem de modelagem precisa ela permite o aperfeiçoamento nos conceitos de modelagem. O seu desenvolvimento foi baseado em técnicas de orientação a objetos. Permite que outras técnicas de modelagem possam ser especificadas utilizando a *UML* como base, permitindo assim que esta linguagem possa ser estendida sem a necessidade de redefinir a sua estrutura interna. No capítulo 3, será apresentada a extensão da *UML* para atender a Engenharia de Sistemas, conhecida como *SysML*.

2.4 RATIONAL UNIFIED PROCESS (RUP)

O *Rational Unified Process*, também conhecido como *RUP* é um modelo de processo de engenharia de software que foi derivado do trabalho sobre a *UML* e do Processo Unificado. Apresenta uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento.

O *RUP* é geralmente descrito a partir de três perspectivas:

1. Dinâmica: mostra as fases do modelo ao longo do tempo.

2. Estática: mostra as atividades realizadas no processo.

3. Prática: sugere boas práticas a serem usadas durante o processo.

Na perspectiva dinâmica, são identificadas quatro fases no processo de software. Segundo Booch, Rumbaugh e Jacobson (2005), uma fase é o período de tempo entre dois importantes marcos de progresso do processo, onde um conjunto de objetivos é alcançado, artefatos são concluídos e decisões são tomadas durante a passagem para a próxima fase.

A seguir seguem as fases do *RUP* ilustradas através da Figura 4.

- Iniciação: refere-se à primeira fase, cujo objetivo principal é definir o escopo do projeto e alinhar a expectativa entre os patrocinadores do projeto sobre o ciclo de vida do projeto.

- Elaboração: a segunda fase visa à criação de uma linha base para a arquitetura do sistema, com objetivo de garantir que a arquitetura e que os requisitos estão estáveis o suficiente para realizar a construção do software, diminuindo os riscos do projeto.

- Construção: a fase de construção é a terceira fase do *RUP* e possui como meta esclarecer todos os requisitos para concluir o desenvolvimento do sistema.

- Transição: a quarta e última fase do *RUP* deve assegurar que o sistema está pronto para ser disponibilizado aos usuários. Neste momento do ciclo de vida

devem ser realizados apenas pequenos ajustes referentes à instalação, à configuração ou aos problemas de usabilidade.

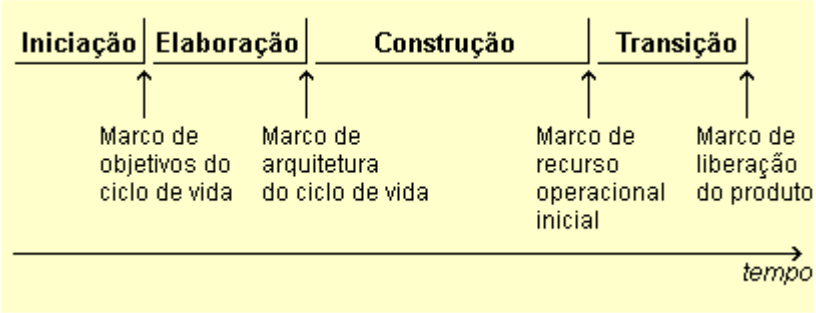

Figura 4 - Fases do RUP e Marcos do Projeto (Rational Unified Process).

As atividades realizadas no processo, também chamadas de disciplinas, representam à perspectiva estática. O *Rational Unified Process* é composto por nove disciplinas:

- Modelagem de negócios: nesta disciplina os processos de negócio são modelados utilizando casos de uso de negócio. Descreve a estrutura e a dinâmica de uma companhia.
- Requisitos: identificação dos requisitos e desenvolvimento de casos de uso para modelar os requisitos de sistema.
- Análise e Design: disciplina responsável pela criação do modelo de projeto. Descreve as diferentes visões da arquitetura do sistema.
- Implementação: os componentes do sistema são implementados e estruturados em subsistemas de implementação. Leva em consideração o teste de unidade e as integrações durante o desenvolvimento do software.

- Teste: disciplina realizada em conjunto com a implementação. Nesta disciplina são criados casos de teste, procedimentos e medidas para acompanhamento dos erros.

- Implantação: nesta disciplina uma versão do produto é criada, distribuída aos usuários e instalada no local de trabalho.

- Gerência de Configuração e Mudança: disciplina que apóia e gerencia as mudanças do sistema. Tem como responsabilidade controlar as modificações e manter a integridade dos artefatos do projeto e das atividades de gerenciamento.

- Gerenciamento de Projeto: disciplina que gerencia o desenvolvimento do sistema, descrevendo estratégias para o trabalho com um processo iterativo.

- Ambiente: refere-se à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento provendo a infra-estrutura necessária para o desenvolvimento do sistema.

Em cada fase ocorrem várias iterações. Uma iteração consiste em um ciclo completo do desenvolvimento, desde o levantamento dos requisitos até uma versão executável de um produto, que constitui um subconjunto do produto final em desenvolvimento. A Figura 5 ilustra o ciclo de vida de desenvolvimento de um software, de acordo com o *RUP*.

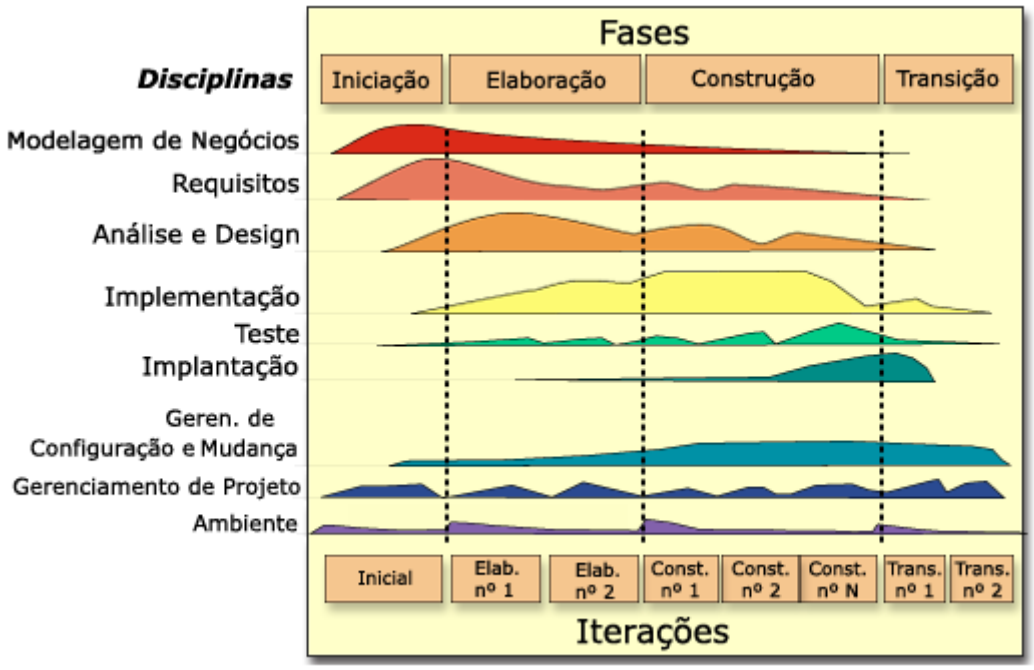

Figura 5 - Ciclo de vida de desenvolvimento de um software (Rational Unified Process).

Na terceira perspectiva do *RUP* são recomendadas boas práticas de engenharia de software para utilização no desenvolvimento de sistemas. Estas boas práticas englobam:

1. Desenvolvimento iterativo;
2. Gerenciamento dos requisitos;
3. Uso de arquiteturas baseadas em componentes;
4. Uso de modelos gráficos para representar o software;
5. Verificação da qualidade do software;
6. Gerenciamento das mudanças do software.

Em resumo, o *RUP* deve ser utilizado considerando-se o ciclo de vida do projeto, a meta do negócio do projeto verificando o escopo e o risco, e o tamanho do esforço de desenvolvimento de software.

2.5 CAPABILITY MATURITY MODEL (CMM)

Para certificar empresas que possuem um processo de desenvolvimento de software definido, ou seja, garantir a estas empresas certo grau de confiança com relação à qualidade do produto ou do serviço entregue, foram criados alguns padrões, tais como: *CMM, SPICE, ISO12207*. Neste trabalho, serão apresentados os conceitos que envolvem o modelo *CMM*, visando apresentar o modelo existente para a disciplina Engenharia de Sistemas.

O *Capability Maturity Model* (Modelo de Maturidade de Capacitação) foi desenvolvido pelo Instituto de Engenharia de Software (*SEI – Software Engineering Institute)* com o objetivo de aumentar as capacitações da indústria de software dos EUA. Este modelo contribuiu para mostrar à comunidade de Engenharia de Software que o aprimoramento dos processos deve ser considerado. O *CMM* pode ser definido como um conjunto de melhores práticas para diagnosticar e avaliar a maturidade do desenvolvimento de softwares nas organizações. Descreve os estágios de maturidade em que as organizações passam durante o ciclo de desenvolvimento do software, realizando avaliação contínua, identificando problemas e ações corretivas, e propondo uma estratégia de melhoria dos processos.

Após o surgimento de diversos modelos de maturidade desenvolvido por outras organizações, o *SEI* desenvolveu o *CMMI (Capability Maturity Model Integration) qu*e se trata de um modelo de capacitação integrada. Este framework representa um modelo de referência que contém as práticas necessárias à maturidade em disciplinas específicas, tais como: *Systems Engineering (SE), Software Engineering (SW), Supplier Sourcing (SS)*, entre outras. Estas disciplinas, também conhecidas como áreas do conhecimento, auxiliam no planejamento da melhoria do processo de toda a organização.

O modelo *CMMI*, segundo o *SEI*, é dividido em duas representações:

- Por estágios: nesta representação, em cada um dos cinco níveis de maturidade são definidos os objetivos que devem ser alcançados pelas organizações. São eles:

    1. Inicial
    2. Gerenciado
    3. Definido
    4. Quantitativamente Gerenciado
    5. Otimização

- Contínuo: nesta representação cada área de processo é avaliada, sendo estabelecido um nível de avaliação de capacitação de 1 a 6 para cada área de processo.

De acordo com Sommerville (2007: 453), o aprimoramento de processos “baseia-se no alcance de um conjunto de objetivos relacionados às boas práticas da engenharia de software e na descrição, padronização e controle de práticas usadas para atingir esses objetivos”.

3 SOLUÇÕES PARA A ENGENHARIA DE SISTEMAS

Conforme foi mencionado no capítulo anterior, os sistemas de automação são considerados sistemas complexos, por envolver diferentes visões e por conter diferentes subsistemas. A aplicação de processos e controles para esse tipo de sistema deve refletir a abrangência do sistema como um todo, e desta forma, a disciplina “Engenharia de Sistemas” destaca-se por focar no desenvolvimento e organização de sistemas grandes e complexos.

O objetivo principal da Engenharia de Sistemas é aumentar a probabilidade de sucesso e diminuir os riscos de falhas. Para atingir esse objetivo essa abordagem considera as necessidades de negócio e as necessidades técnicas de todos os clientes (interessados) no sistema, visando prover qualidade do produto e atingir as reais necessidades dos usuários.

Visando atender a esta crescente demanda, ferramentas foram desenvolvidas para suportar as características específicas de sistemas complexos, tais como os sistemas de automação. Neste capítulo serão apresentadas algumas destas ferramentas e soluções voltadas para a engenharia de sistemas. Também será estudado o funcionamento do processo da engenharia de sistemas para melhor compreensão das ferramentas.

3.1 O PROCESSO DE ENGENHARIA DE SISTEMAS

De acordo com o *Defense Acquisition University Press,* o processo de engenharia de sistemas possibilita a transformação das necessidades e dos requisitos em um conjunto de produtos de sistema e descrições de processo, gerando informações para tomada de decisões e provendo entradas para o próximo nível de desenvolvimento. O processo é descrito através de:

- Entradas: necessidades dos clientes, objetivos, requisitos e restrições de projeto.
- Análise de requisitos: passo inicial do processo para analisar as entradas e definir os requisitos funcionais e as restrições de projeto.
- Análise funcional e alocações: representa a arquitetura funcional do produto. Permite o melhor entendimento sobre o que o sistema deve fazer e quais maneiras de realizá-lo.
- Síntese de projeto: define o produto em termos de elementos físicos e softwares. Referenciado como a arquitetura física.
- Verificação: confirma que a síntese de projeto resulta na arquitetura física que atende aos requisitos de sistema.
- Análise de sistemas e controle: atividades de gerenciamento requeridas para medir o progresso, avaliar alternativas, e documentar dados e decisões do projeto.
- Saídas: processo dependente dos níveis de desenvolvimento.

O processo é aplicado sequencialmente, sendo um nível por vez adicionando detalhes e definições do sistema a cada nível de desenvolvimento. A Figura 6 ilustra esse processo, podendo-se visualizar os elementos que o compõem, conforme foi descrito acima.

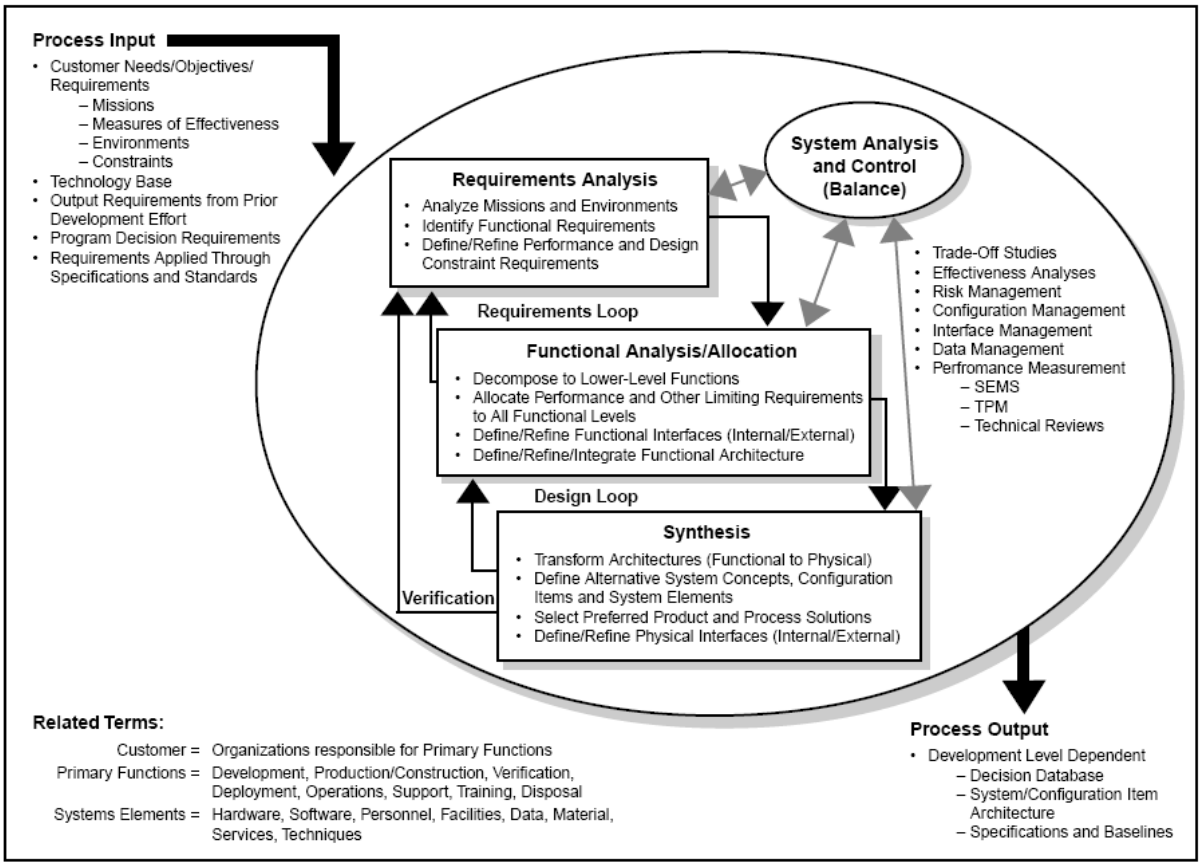

Figura 6 - O Processo de Engenharia de Sistemas (DoD, 2001)

No processo de Engenharia de Sistemas é importante observar que existe um aumento do nível de descrição dos detalhes do produto e do processo a cada aplicação. A saída de cada aplicação é a entrada para a próxima aplicação do processo.

Segundo Sommerville (2007), o processo de engenharia de sistemas foi uma influência no modelo “Cascata” do processo de desenvolvimento de software. As fases deste processo são ilustradas na Figura 7.

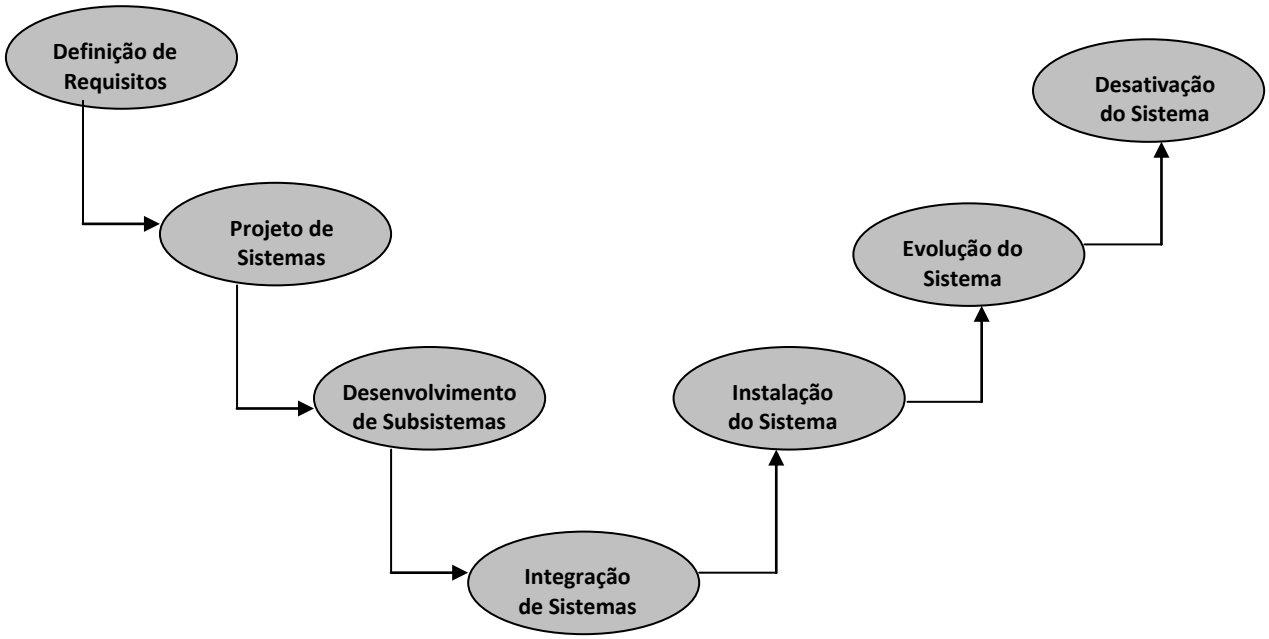

Figura 7 – Processo de Engenharia de Sistemas (Sommerville, 2007, p.17).

Através deste processo, nota-se que existem importantes diferenças entre o processo de engenharia de sistemas e o de desenvolvimento de software. Destacam-se o escopo limitado para re-trabalho durante o desenvolvimento do sistema, pois não é tão simples realizar alterações em decisões de engenharia de sistemas como é possível nos softwares em desenvolvimento. Outra característica é o envolvimento interdisciplinar, ou seja, muitas disciplinas de engenharia podem ser envolvidas. Por exemplo, na equipe de engenharia de sistemas para um sistema de automação dos processos de transferência e estocagem de produtos de uma refinaria, podemos considerar as seguintes disciplinas: Engenharia de Software,

Engenharia de Produção, Engenharia Elétrica, Engenharia Eletrônica, Engenharia Mecânica, entre outras.

A seguir, será apresentado o *RUP-SE* que permite a implantação de um processo para a engenharia de sistemas visando a simplificação de atividades relacionadas ao ciclo de vida de desenvolvimento e dando suporte às necessidades da engenharia de sistemas.

3.2 RATIONAL UNIFIED PROCESS FOR SYSTEMS ENGINEERING (RUP-SE)

O *RUP-SE* é o *plug-in* do *RUP* para a Engenharia de Sistemas, desenvolvido para suportar o desenvolvimento de sistemas em grande escala. Complementa o *RUP* com artefatos adicionais, novas atividades e funções que apóiam a criação desses artefatos. Podemos considerar que o *RUP* sem o *plugin* do *RUP-SE* é um processo para a engenharia de software. As seções a seguir apresentam algumas características do *RUP-SE.*

3.2.1 Framework de Arquitetura do *RUP-SE*

O *RUP-SE p*ossui um modelo de dimensões que permite separar os interesses de diferentes times envolvidos no *design* e na construção do sistema. As

duas dimensões consideradas são: os pontos de vista (*viewpoints)* e os níveis de especificação do modelo (*models)*.

Inclui uma arquitetura de sistema que permite considerar diferentes perspectivas para expressar a arquitetura a partir dos vários pontos de vista (empresa, computação ou lógico, informação, engenharia ou físico, processo) e dos níveis de especificação. Utilizando como base o *Rational Unified Process,* os quatro níveis arquiteturais são descritos abaixo:

- Contexto: necessário para expressar o sistema e seus atores.
- Análise: realizar a análise inicial do sistema para estabelecer a abordagem.
- Design: realização do modelo de análise para *hardware, software* e pessoas.
- Implementação: realização do modelo de *Design* em configurações específicas.

Um conjunto de diagramas é utilizado para capturar a arquitetura do sistema através dos diferentes pontos de vista e dos níveis de especificação, conforme ilustra a Tabela 2.

Models	Viewpoints				
	Enterprise	Computation	Information	Engineering	Process
Context	UML organization model System context diagram	Enterprise object model	Enterprise data model	Enterprise locality (Distribution of enterprise resources)	
Analysis		Subsystem diagram	System data model	System locality diagram	System Process diagram
Design	Business Worker Survey	Subsystem class model Software component model	System data schema	Descriptor node diagram	Detailed process
Implementation	Worker Instructions	Configurations: deployment diagram with software system components			

Tabela 2 - Pontos de vista x níveis de especificação. (RUP ® SE1.1).

Praticamente todos os artefatos especificados nesta tabela são baseados em diagramas *UML,* seguindo o modelo de arquitetura do RUP 4+1. No entanto, não é possível capturar todos os interesses através desse modelo, sendo necessário estender o modelo de arquitetura do RUP 4+1 para a estrutura do modelo *RUP-SE* para considerar novas visualizações. Por exemplo, através da Tabela 2, verificamos que no nível de análise, para o ponto de vista de Engenharia utiliza-se o diagrama de localidade (*Locality Diagram*). Este diagrama captura de forma fictícia onde o processamento ocorre, amarrando a localização do processamento a uma localização geográfica específica. Os diagramas *UML* específicos para suportar a arquitetura do sistema serão estudados mais adiante ao apresentar a extensão da *UML* para a engenharia de sistemas, conhecida como *SysML*.

3.2.2 Gerenciamento

Existem poucas diferenças entre o gerenciamento de projetos de engenharia de sistemas e o gerenciamento de projetos de desenvolvimento de software baseados no *RUP.* As principais diferenças são para englobar as novas funções e atividades na engenharia de sistemas, necessárias devido ao tamanho e complexidade do projeto que em sua maioria são compostos por sub-sistemas.

De acordo com o modelo proposto no *RUP-SE*, a organização de um projeto de engenharia de software é realizada através da junção de equipes de desenvolvimento, onde cada equipe é responsável por funções específicas. A estrutura de um projeto *RUP-SE* é exibida na Figura 8 demonstrando o relacionamento entre as equipes.

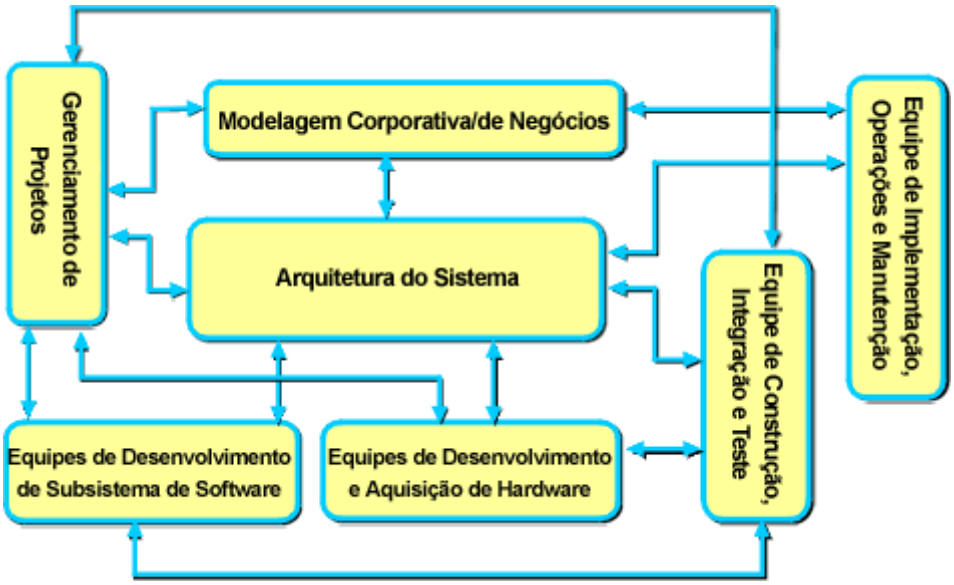

Figura 8 - Organização de um projeto de Engenharia de Sistemas (Plug-in RUP-SE).

- **Modelagem Corporativa/de Negócios:** equipe responsável por analisar a necessidade do negócio.

- **Arquitetura do Sistema:** equipe responsável por determinar o contexto do sistema, derivando os requisitos do sistema. Desenvolve as visões de Subsistemas e Localidades.

- **Gerenciamento de Projeto:** equipe responsável por desempenhar a função de coordenador de projeto.

- **Desenvolvimento de Subsistemas de Software:** equipe responsável pelo desenho e implementação dos subsistemas.

- **Desenvolvimento e Aquisição de Hardware:** equipe responsável pelo desenho, especificação e entrega dos componentes de *hardware.*

- **Construção, Integração e Teste:** equipe responsável por receber os componentes de *hardware,* os códigos das equipes de desenvolvimento e construir e instalar os componentes de *software* e *hardware* em um ambiente para que possam ser realizados os testes no sistema.

- **Implementação, Operações e Manutenção:** equipe responsável por tratar os problemas operacionais, servindo de canal de comunicação com os usuários.

3.2.3 Análise de Requisitos

Seguindo os conceitos do *RUP* e da *UML*, o *RUP-SE* trabalha com dois tipos que artefatos para representar os requisitos do sistema: Casos de Uso e Especificação Suplementar.

O *RUP-SE* também trabalha com o conceito de “Requisitos Derivados”, que são os requisitos determinados a partir da colaboração dos elementos arquiteturais, ou seja, um requisito derivado é um requisito que surge após o refinamento de um requisito de usuário.

Também trabalha com o conceito de “Requisitos Alocados”, que são os requisitos que foram designados aos componentes do sistema, como por exemplo, subsistema de hardware e software. Neste caso, o requisito de sistema é atribuído para um elemento arquitetural.

Disponibiliza um padrão de processo para derivar os requisitos a partir de elementos arquiteturais. Neste processo, a especificação do sistema deve definir os casos de uso, os serviços de sistema e os requisitos suplementares.

Uma técnica utilizada para derivar os requisitos funcionais para os elementos de análise é o fluxo de casos de uso. Esta técnica pode ser aplicada para adicionar detalhes dentro de um nível de modelo e especificar elementos em um nível mais baixo. Casos de uso descrevem como entidades e elementos, num determinado contexto, colaboram para atingir algum objetivo.

Os requisitos suplementares são capturados como atributos das classes do sistema. Como parte do processo de análise, o arquiteto da equipe de

desenvolvimento cria a versão inicial do Diagrama de Localidade (*Locality Diagram)*. Este diagrama faz parte do ponto de vista de Engenharia no nível de análise, conforme ilustrado na Tabela 2. Fornece um contexto para tratamento dos requisitos não-funcionais.

Os diagramas de localidade documentam diferentes abordagens de engenharia para uma empresa. Possuem dois elementos principais: localidade e conexões. A Figura 9 exemplifica este diagrama.

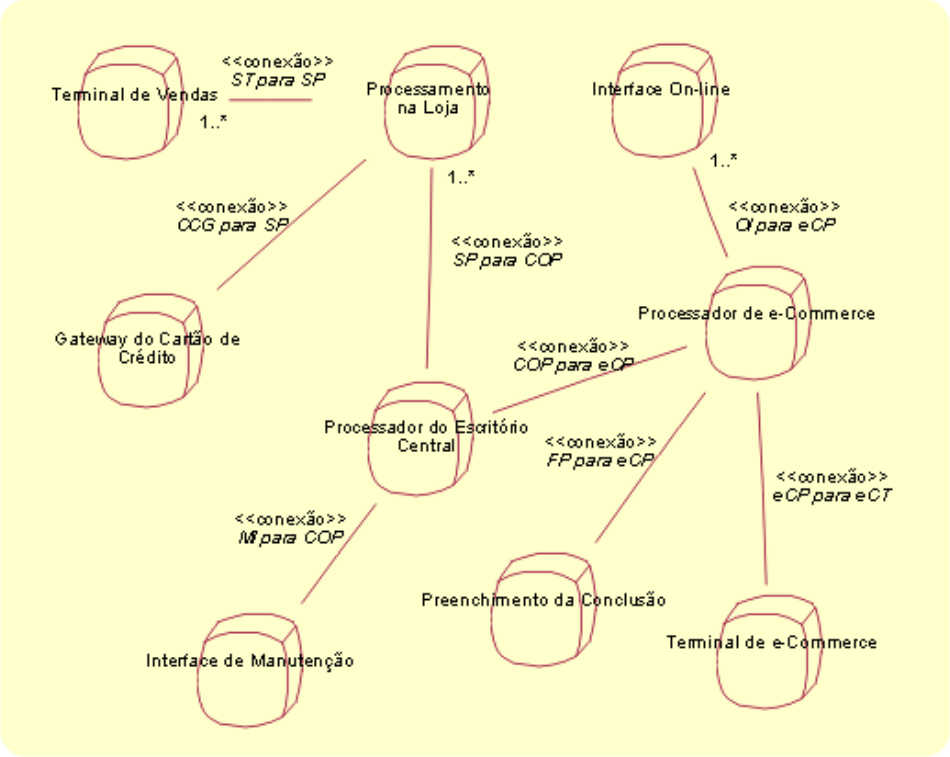

Figura 9- Exemplo de Diagrama de Localidade. (Plug-in RUP-SE)

Outros exemplos de diagramas serão apresentados na seção 3.3 - SYSTEMS MODELLING LANGUAGE (SysML).

3.2.4 Alterações no *RUP*

Com base no IBM Rational Unified Process (RUP), as principais alterações realizadas no RUP para suportar as necessidades da engenharia de sistemas são apresentadas abaixo, destacando-as por disciplinas (os novos elementos são indicados com asterisco):

- **Modelagem de Negócios:** não há alterações.
- **Requisitos:** nesta disciplina as técnicas para elicitação de requisitos são semelhantes tanto no *RUP* quanto no *RUP-SE.* Entretanto como no contexto de sistemas temos elementos como softwares, hardwares, humanos, entre outros, existem algumas restrições ou características físicas que devem ser consideradas na especificação. Desta forma ganham muita importância os requisitos não-funcionais. Para atender as necessidades da engenharia de sistemas novos artefatos foram criados e alguns foram alterados. Algumas atividades também foram alteradas, conforme ilustra a Tabela 3.

Disciplina	Atividade	Tarefa	Artefato	Função
Requisitos	Gerenciar Requisitos Variáveis	Definir o Contexto do Sistema*	Caso de Uso Visão Especificações Suplementares Modelo de Caso de UsoEspecificação de Requisitos do Sistema*	Analista de Sistemas Especificados de Requisitos
	Gerenciar o Escopo do Sistema	Priorizar Casos de Uso do Sistema* Desenvolver a Visão		
	Refinar a Definição do Sistema	Definir o Contexto do Sistema* Detalhas os Requisitos do Sistema* Detalhar um Caso de Uso		
	Definir o Sistema	Localizar Agentes e Casos de Uso Definir o Contexto do Sistema* Desenvolver a Visão		

Tabela 3 - Novos artefatos da disciplina de Requisitos. (Plug-in RUP-SE)

- **Análise e Design:** nesta disciplina existe um conjunto de tarefas descritas separadamente do fluxo padrão da análise e do design, com o objetivo de produzir uma prova de conceito arquitetural, definir a estrutura lógica ou física do sistema e de refinar a arquitetura. Algumas atividades de nível mais baixo passam a fazer parte de especificações para subsistemas. A Tabela 4 apresenta as alterações nesta disciplina.

Disciplina	Atividade	Tarefa	Artefato	Função
Análise & Design - Engenharia de Sistemas	Sintetizar Arquitetura do Sistema*	Análise Arquitetural do Sistema* Construir Prova de Conceito da Arquitetura do Sistema* Avaliar a Viabilidade da Prova de Conceito da Arquitetura do Sistema*	Requisitos Suplementares do Subsistema* Modelo de Caso de Uso do Subsistema* Visão do Subsistema* Modelo de Análise do Sistema* Documento de Arquitetura do Sistema* Prova de Conceito da Arquitetura do Sistema* Modelo de Implementação do Sistema* Modelo de Design do Sistema* Realização de Caso de Uso do Sistema* Operação do Subsistema* Realização da Operação do Subsistema*	Arquiteto do Sistema* Designer do Sistema*
	Definir Arquitetura do Sistema Candidato*	Análise Arquitetural do Sistema* Análise da Operação*		
	Analisar Comportamento do Sistema*	Design da Operação* Refinar Estrutura do Sistema* Refinar Processos do Sistema* Refinar Modelo de Implementação do Sistema*		
	Refinar Arquitetura do Sistema*	Análise Arquitetural do Sistema* Refinar Estrutura do Sistema* Refinar Processos do Sistema* Refinar Modelo de Implementação do Sistema*		

Tabela 4 - Novos artefatos da disciplina Análise e Design. (Plug-in RUP-SE)

- **Implementação:** não há alterações.
- **Teste:** não há alterações.
- **Implantação:** não há alterações.
- **Gerenciamento de Configuração e Mudança:** não há alterações.
- **Gerenciamento de Projeto:** nesta disciplina algumas tarefas de planejamento e gerenciamento ficam mais complexas, pois é necessário que o planejamento ocorra em vários níveis sincronizando com os níveis inferiores (planejamento dos subsistemas). A Tabela 5 apresenta as alterações.

Disciplina	Atividade	Tarefa	Artefato	Função
Gerenciamento de Projeto	Desenvolver Planos	Revisão de Planejamento do Projeto Compilar Planos Fases e Iterações do Plano Definir Processos de Monitoramento e Controle Definir a Equipe e a Organização do Projeto	Plano de Iteração Plano de Desenvolvimento do Sistema*	Coordenador de Projeto Revisor de Projeto
	Conceber Novo Projeto	Iniciar Projeto		
	Gerenciar Iteração	Adquirir Equipe Iniciar Iteração Avaliar Iteração		
	Monitorar e Controlar Projeto	Monitorar Status do Projeto Planejar e Designar Trabalho Resolver Exceções e Problemas		
	Planejar Próxima Iteração	Desenvolver Plano de Iteração		
	Finalizar Fase	Preparar para Finalizar Fase Revisão do Marco de Ciclo de Vida		
	Finalizar Projeto	Preparar para Finalizar Projeto Revisão de Aceitação do Projeto		

Tabela 5 - Novos artefatos da disciplina Gerenciamento de Projetos. (Plug-in RUP-SE)

- **Ambiente:** não há alterações.

Com as modificações de itens existentes e inclusão de novos itens (artefatos, tarefas...), o *plug-in RUP-SE* possibilita fornecer o suporte para o escopo mais amplo e as preocupações adicionais da Engenharia de Sistemas.

3.2.5 IBM Rational Method Composer

O *IBM Rational Method Composer* é uma plataforma de ferramentas que permite a implementação, o desenvolvimento ou a manutenção dos processos existentes nas organizações. Esta ferramenta de engenharia de processos, além de ser um gerenciador de conteúdo, visa entre outros objetivos:

- Prover uma base de conhecimento que possa ser gerenciada e ter seu conteúdo publicado em *HTML;*
- Disponibilizar recursos de engenharia do processo, fornecendo catálogos de processos pré-definidos para situações típicas de projetos mas que podem ser personalizadas de acordo com as necessidades individuais.

O *Rational Method Composer* é independente do *RUP*, pois por ser uma ferramenta de objetivo geral fornece suporte para a maioria dos modelos do ciclo de vida do desenvolvimento de um produto.

Tem como princípio a separação de conteúdo de método em processos, onde o conteúdo de método preocupa-se em descrever como as metas de

desenvolvimento são atingidas e os processos descrevem o ciclo de vida de desenvolvimento. A Figura 10 ilustra como essa separação é realizada.

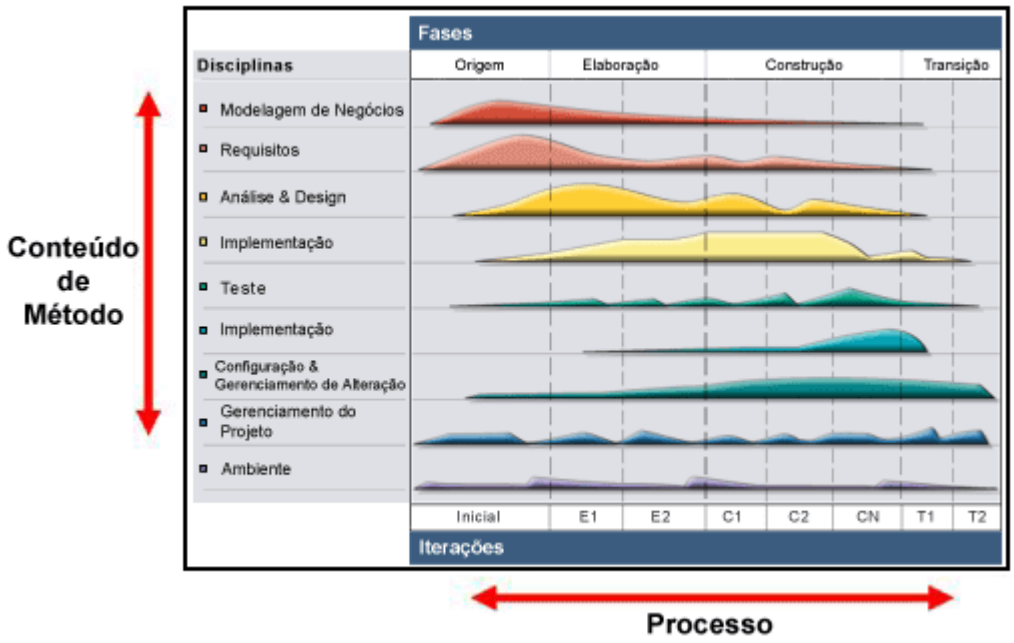

Figura 10 - Conteúdo de Método x Processo (Ferramenta IBM Rational Method Composer)

A ferramenta trabalha com perspectivas, que são utilizadas para diferentes operações. As perspectivas mais utilizadas são: Autoria e Navegação.

A Figura 11 apresenta as opções disponíveis para a perspectiva de Autoria. Nesta perspectiva é possível criar e manter conteúdos de métodos utilizando os editores disponíveis.

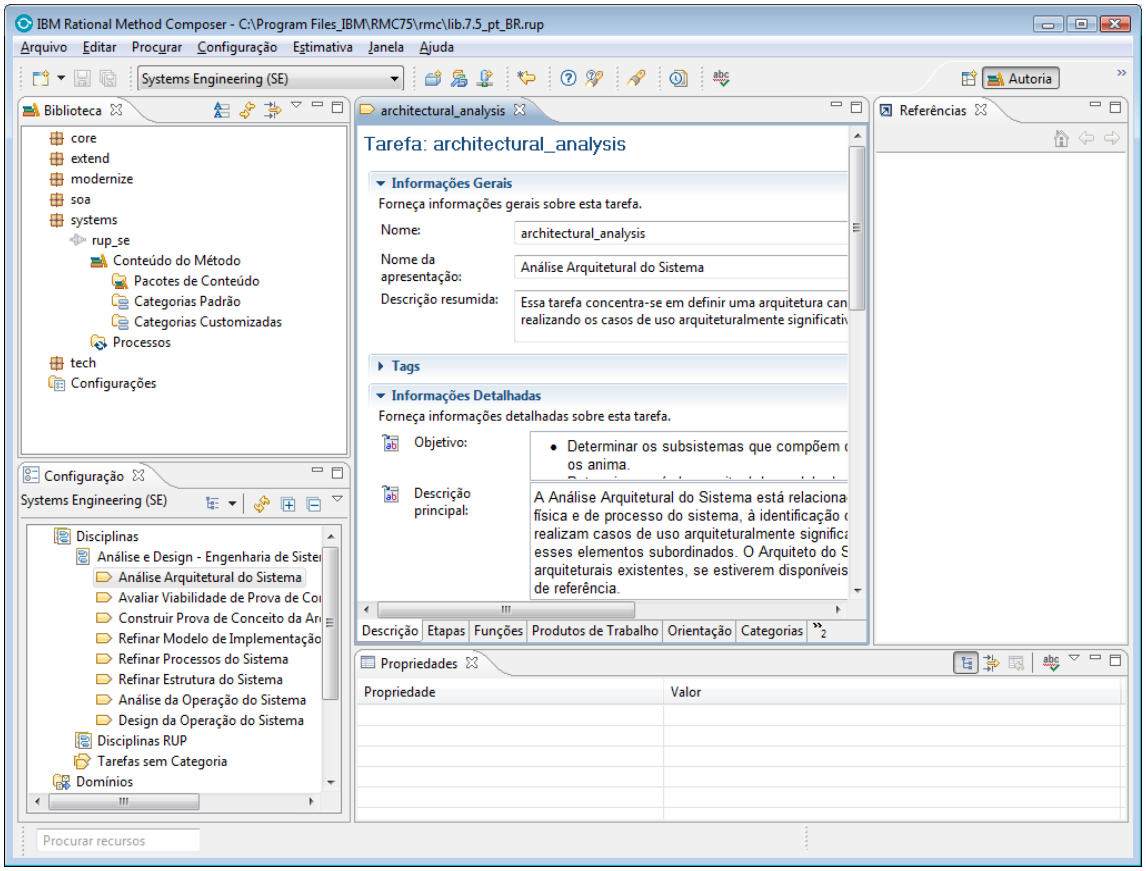

Figura 11 - Perspectiva de Autoria

A Figura 12 ilustra as opções disponíveis para a perspectiva de Navegação. Esta perspectiva fornece uma visualização mais completa da configuração que foi gerada.

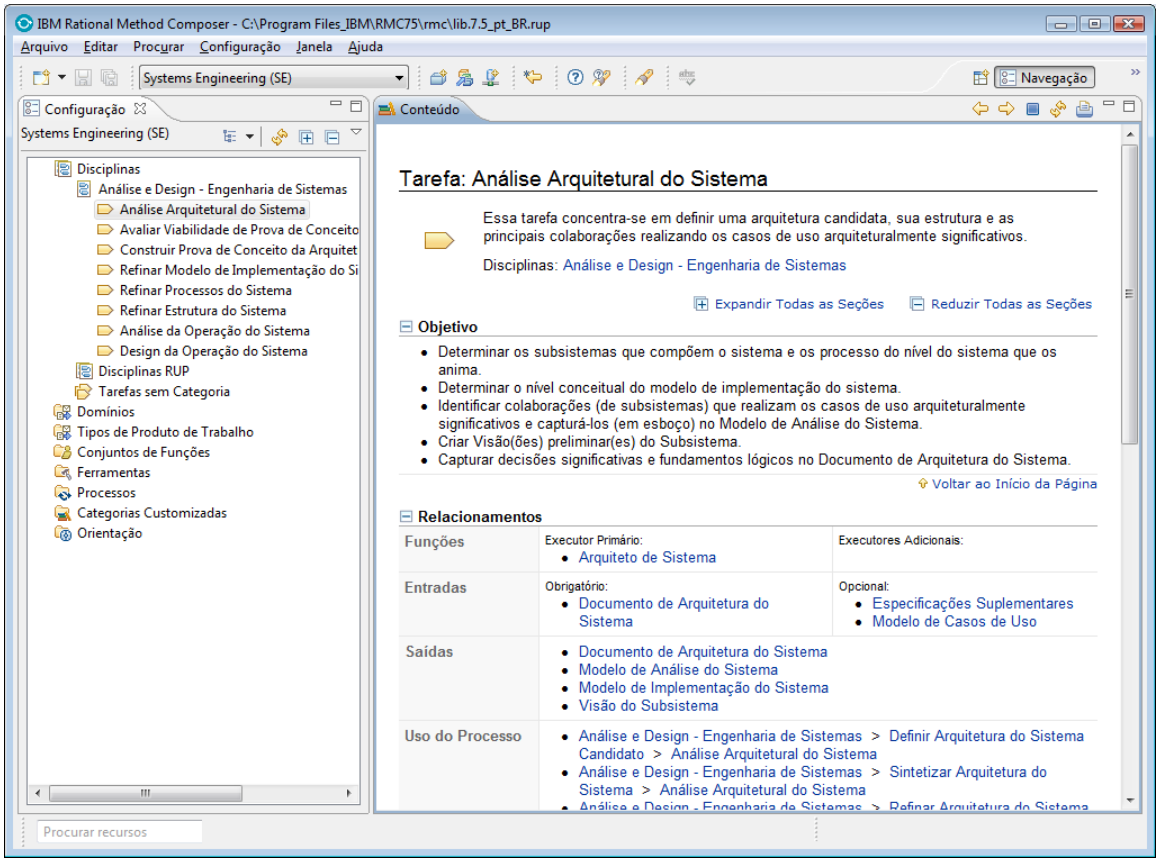

Figura 12 - Perspectiva de Navegação

A orientação de processo será gerada utilizando a Biblioteca de Métodos, que trata-se de um repositório de elementos organizada em um conjunto de *plug-ins de métodos.* Cada *plug-in* possui seu conteúdo separado em conteúdo de método e processos.

Uma Configuração de Método é construída através de elementos da Biblioteca de Métodos, ou seja, após criar uma configuração, o conteúdo na visualização conterá apenas os elementos da biblioteca utilizados na configuração.

Um dos principais benefícios da utilização desta ferramenta é a possibilidade de selecionar uma configuração base do *RUP* (para projeto pequeno, projeto médio, projeto grande, engenharia de sistemas, arquitetura orientada a serviços, entre

outros), permitindo adicionar *plug-ins* e componentes de processos relevantes a cada projeto.

3.3 SYSTEMS MODELLING LANGUAGE (SysML)

A *SysML (Systems Modeling Language)* é uma linguagem padrão de modelagem que representa uma customização da *UML (Unified Modeling Language)* para a Engenharia de Sistemas, desenvolvida pelo *OMG (Object Management Group)*. O objetivo desta customização visa apoiar a modelagem de uma ampla quantidade de sistemas que podem incluir hardware, software, dados, pessoal, procedimentos e facilidades.

É definida como um subconjunto da *UML 2.0* contendo as extensões necessárias para satisfazer os requisitos da *UML* para a Engenharia de Sistemas. Desta forma, reutiliza alguns diagramas (casos de uso, máquina de estados, entre outros) e adiciona dois novos diagramas (*Requirements e Parametrics).* A Figura 13 apresenta os tipos de diagramas da *SysML* e a Figura 14 ilustra o relacionamento entre *SysML* e *UML.*

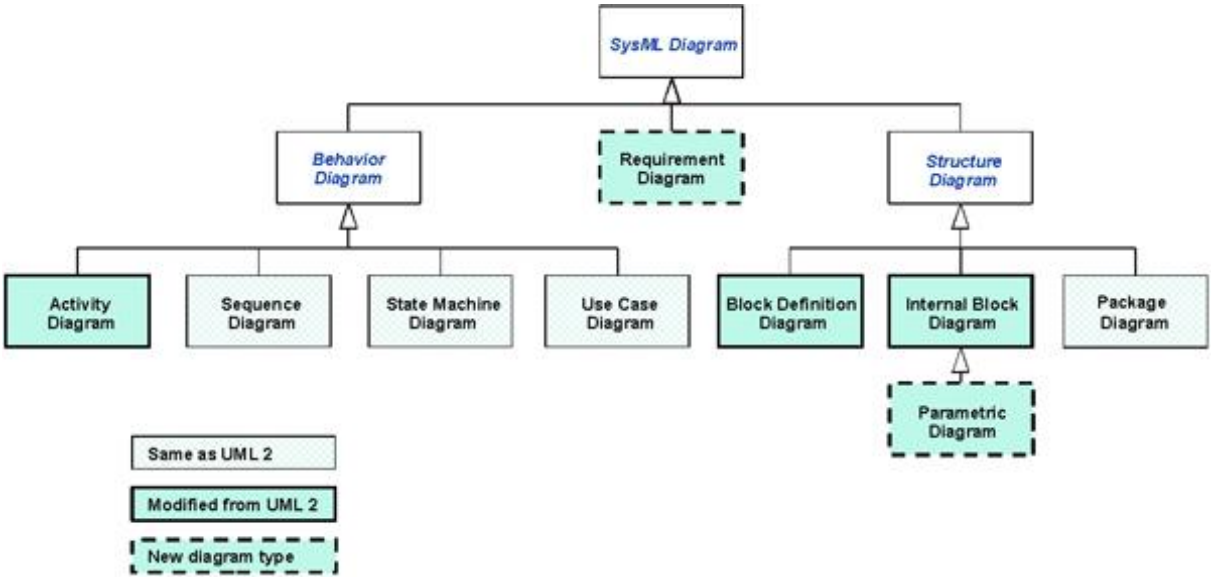

Figura 13 - Tipos de diagramas da SysML (OMG Systems Modeling Language).

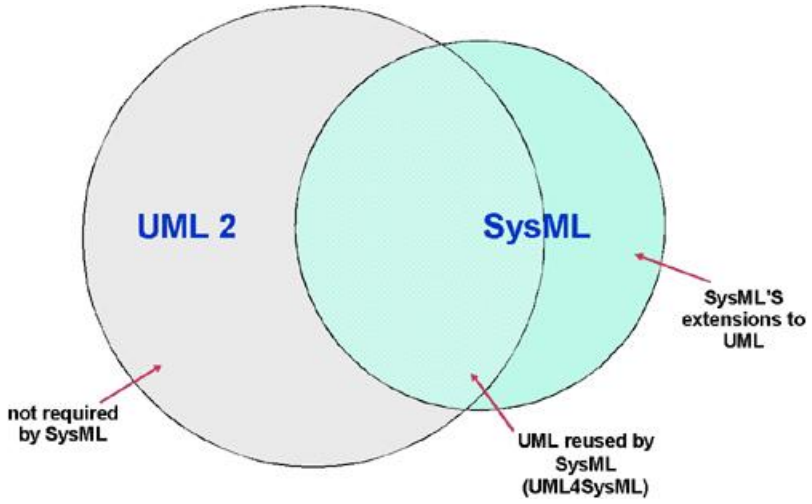

Figura 14 - Relacionamento entre SysML e UML (OMG Systems Modeling Language).

Os diagramas que compõem a *SysML* são agrupados por contexto, conforme ilustrado na Figura 13, sendo divididos em:

- Diagramas de Estrutura;
- Diagramas de Comportamento;

- Diagrama de Requisitos;
- Diagrama Paramétrico.

Os Diagramas de Estrutura representam as construções estáticas e estruturais utilizadas na *SysML* e incluem o **Diagrama de Pacotes** (utilizado para organizar o modelo), o **Diagrama de Definição de Blocos** (define as características dos blocos e os relacionamentos entre os blocos, tais como: associações, generalizações e dependências. É equivalente ao Diagrama de Classes da *UML*) e o **Diagrama de Blocos Interno** (captura a estrutura interna de um bloco, tais como suas propriedades e operações).

Os blocos, presentes nos diagramas de estrutura, são elementos utilizados para representar todos os tipos de componentes de sistemas (Ex: funcionais, físicos ou humanos) com o objetivo de descrever a estrutura destes componentes. A Figura 15 ilustra um Diagrama de Definição de Blocos e a Figura 16 apresenta um Diagrama de Blocos Interno.

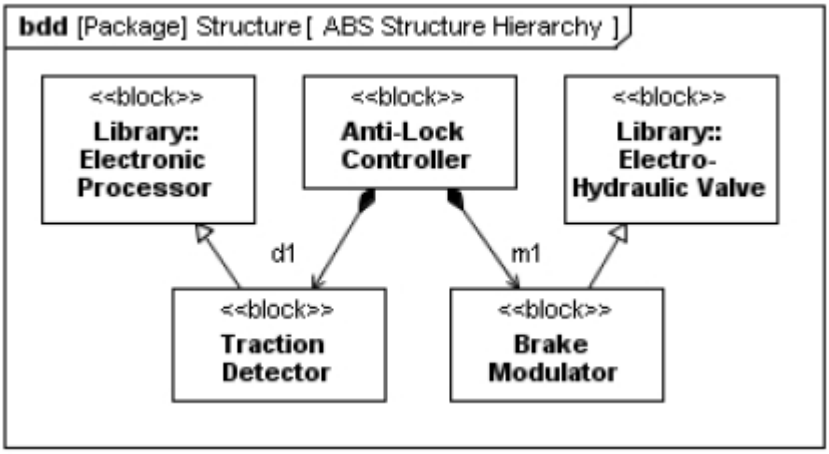

Figura 15 - Diagrama de Definição de Blocos (OMG SysML tutorial. Disponível em: http://www.omgsysml.org)

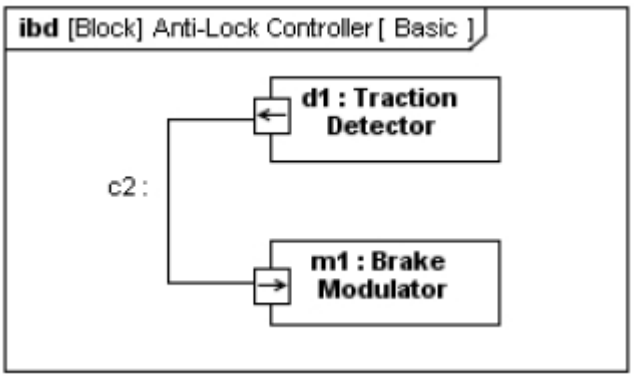

Figura 16 - Diagrama de Blocos Interno (OMG SysML tutorial. Disponível em: http://www.omgsysml.org)

Os Diagramas de Comportamento representam as construções comportamentais e dinâmicas na *SysML*. Incluem o **Diagrama de Casos de Uso** (fornece uma descrição de alto nível das funcionalidades através da interação dos sistemas e partes do sistema), o **Diagrama de Atividades** (representa o fluxo de dados e controle entre as atividades), o **Diagrama de Sequência** (representa a interação entre as partes colaboradoras do sistema) e o **Diagrama de Máquina de Estados** (descrevem as transições de estado e as ações que um sistema ou partes de um sistema realizam em resposta aos eventos). Para a Engenharia de Sistemas, o Diagrama de Atividades é um dos mais interessantes, pois foca em quais atividades necessitam ser realizadas, em qual ordem e quais as entradas das atividades, em vez de focar em quais entidades realizam quais tarefas. Este diagrama destaca a dependência de entradas e saídas e corresponde para a Engenharia de Sistemas em um fluxo funcional. A Figura 17 ilustra um Diagrama de Atividades.

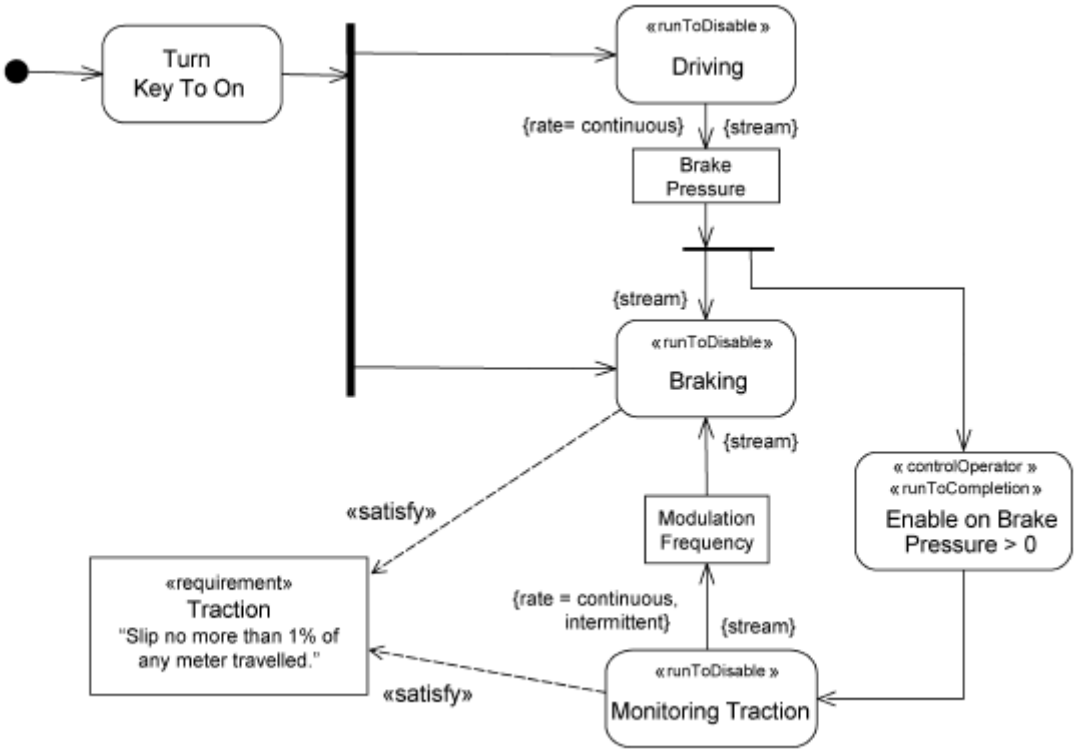

Figura 17 - Diagrama de Atividades (BOCK,2005)

O Diagrama de Requisitos (*Requirement Diagram) permite* a representação dos requisitos como elementos do modelo. Os requisitos descrevem as funcionalidades do produto, assim como as restrições em que as funcionalidades devem ser realizadas. Sendo representados como elementos do modelo tornam-se parte integral da arquitetura do produto. Um diagrama de requisitos pode conter tanto os requisitos funcionais quanto os não-funcionais. A Figura 18 apresenta um exemplo do Diagrama de Requisitos.

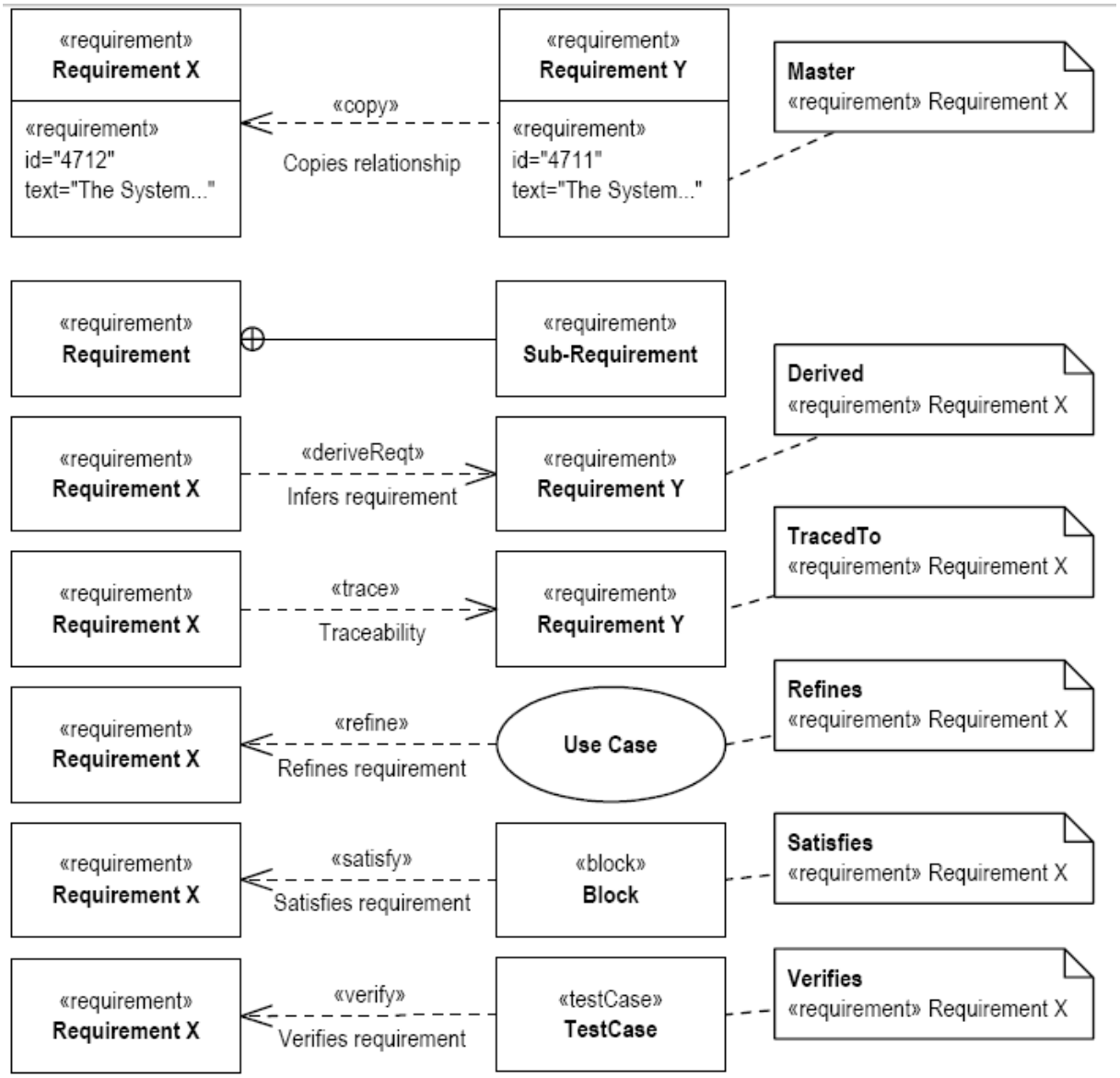

Figura 18 - Diagrama de Requisitos (OMG Systems Modeling Language)

O Diagrama Paramétrico *(Parametric Diagram)* apresenta as restrições paramétricas entre os elementos estruturais. Utilizado para verificar o desempenho e realizar a análise quantitativa. Um modelo paramétrico descreve as restrições entre as propriedades de um sistema e geralmente tais restrições são expressas através de equações matemáticas. Ao contrário da *UML, a SysML* permite o reuso de equações através das relações paramétricas. O reuso de equações é um requisito crítico para a SysML. A Figura 19 ilustra um exemplo do Diagrama Paramétrico.

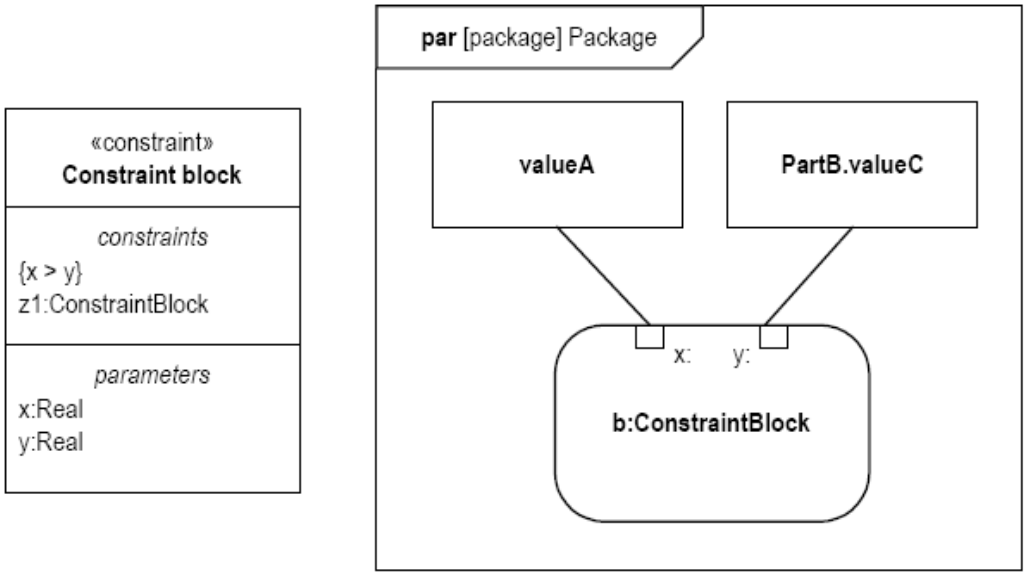

Figura 19 - Diagrama Paramétrico (OMG Systems Modeling Language)

A Tabela 6 apresenta uma comparação entre os diagramas *SysML* com os diagramas análogos da *UML.*

SYSML DIAGRAM	PURPOSE	UML DIAGRAM ANALOG
Activity diagram	Show system behavior as control and data flows. Useful for functional analysis. Compare Extended Functional Flow Block diagrams (EFFBDs), already commonly used among systems engineers.	Activity diagram
Block Definition diagram	Show system structure as components along with their properties, operations and relationships. Useful for system analysis and design.	Class diagram
Internal Block diagram	Show the internal structures of components, including their parts and connectors. Useful for system analysis and design.	Composite Structure diagram
Package diagram	Show how a model is organized into packages, views and viewpoints. Useful for model management.	Package diagram
Parametric diagram	Show parametric constraints between structural elements. Useful for performance and quantitative analysis.	N/A

Requirement diagram	Show system requirements and their relationships with other elements. Useful for requirements engineering.	N/A
Sequence diagram	Show system behavior as interactions between system components. Useful for system analysis and design.	Sequence diagram
State Machine diagram	Show system behavior as sequences of states that a component or interaction experience in response to events. Useful for system design and simulation/code generation.	State Machine diagram
Use Case diagram	Show system functional requirements as transactions that are meaningful to system users. Useful for specifying functional requirements. (Note potential overlap with Requirement diagrams.)	Use Case diagram
Allocation tables* *dynamically derived tables, not really a diagram type	Show various kinds of allocations (e.g., requirement allocation, functional allocation, structural allocation). Useful for facilitating automated verification and validation (V&V) and gap analysis.	N/A
N/A		Component diagram
N/A		Communication diagram
N/A		Deployment diagram
N/A		Interaction overview diagram
N/A *the inability to define Block instances in SysML is frequently cited as a serious design flaw of SysML		Object diagram
N/A		Timing diagram

Tabela 6 - Comparação diagramas SysML x UML (SysML Fórum)

De acordo com a OMG SysML™ , a *SysML* possui alguns princípios fundamentais. São eles:

- Procurar satisfazer aos requisitos da *UML* para a Engenharia de Sistemas;
- Reutilizar a *UML;*

- Extender a *UML* como necessidade de atender aos requisitos para a Engenharia de Sistemas;
- Permitir o particionamento do modelo de elementos em grupos lógicos para minimizar as dependências;
- Especificar os pacotes *SysML* como uma extensão do metamodelo *UML;*
- Suportar a interoperabilidade entre outras ferramentas.

Em resumo, a *SysML* procura proporcionar aos engenheiros de sistemas uma modelagem específica para a especificação de sistemas complexos, que incluem além do software, elementos como hardware, processos, informações, pessoal, entre outros. Como a *UML* preocupa-se apenas com o software, não consegue satisfazer as necessidades da engenharia de sistemas, existindo desta forma uma linguagem que permite apoiar atividades fundamentais, como a engenharia de requisitos ou as restrições paramétricas. Essas atividades são consideradas recursos essenciais para a engenharia de sistemas.

3.4 SYSTEMS ENGINEERING CAPABILITY MATURITY MODEL

Podemos considerar que, uma das formas de verificar se um processo de desenvolvimento está sendo aplicado corretamente e se este processo está trazendo benefícios, é medir esse processo. Para isso existem ferramentas específicas que auxiliam empresas a medirem e aprimorarem seus processos.

Existem dois principais modelos de processos para a engenharia de sistemas:

- *SE-CMM (Systems Engineering Capability Maturity Model)*: modelo desenvolvido pelo *SEI (Software Engineering Institute).*
- *SECAM (Systems Engineering Capability Assessment Model):* modelo desenvolvido pelo *INCOSE (International Council on Systems Engineering)*.

Neste trabalho será apresentado o modelo desenvolvido pelo *SEI*, o *SE-CMM*. Este modelo foi incluído no *framework CMMI* de aprimoramento de processos. Também serão discutidas algumas diferenças entre os modelos mencionados neste trabalho.

O *SE-CMM (Systems Engineering Capability Maturity Model)* é uma ferramenta desenvolvida para ser utilizada como um guia para processos de desenvolvimento, sendo eles documentados ou não-documentados. Ou para medir os processos implementados nas empresas e determinar os pontos de melhoria. Basicamente trata-se de um roteiro para a engenharia de sistemas, descrevendo as atividades principais que uma empresa deveria realizar para obter melhorias no gerenciamento de seus processos.

As atividades principais são chamadas de “Áreas de Processos”. A Tabela 7 apresenta as 18 áreas de processos que fazem parte do modelo *SE-CMM.*

Process Area Title	Process Area Description
Analyze Candidate Solutions	Perform studies and analyses which result in the selection of a solution to meet the specified constraints.
Derive and Allocate Requirements	Analyze the sys. & other req. & derive a more detailed & precise set of req, allocate to sys. fnct., people, supporting processes, products & services, which can be used to synthesize soln.
Evolve System Architecture	Transform the functional architecture into the physical architecture for the system and evaluate the impact of design decisions on life cycle costs, manufacturability, and supportability
Integrate Disciplines	Identify those disciplines necessary for effective system development and create an environment in which they can work together effectively toward a common agenda.
Integrate System	Ensure the system elements will function as a whole.
Understand Customer Needs & Expectations	Elicit, stimulate, analyze, & communicate customer needs & expectations to obtain better understanding of what will satisfy the customer.
Verify and Validate System	Ensure that the team performs increasingly comprehensive evaluations to ensure that evolving work products will meet all requirements.
Ensure Quality	Address not only the quality of the system, but also the quality of the process being used to create the system and the degree to which the project follows the defined process.
Manage Configurations	Maintain data and status of identified configuration units, and analyze and control changes to the system and its configuration units.
Manage Risk	ID, assess, monitor, & mitigate risks to the success of the SE activities the overall tech. effort.
Mntr & Cntrl Tech Efrt.	Provide adequate visibility into actual progress and risks.
Plan Technical Effort	Establish plans that provide the basis for scheduling, costing, controlling, tracking and negotiating the nature and scope of the technical work.
Define Org. SE Process	Create and manage organization's standard systems engineering processes.
Improve Org. SE Processes	Gain competitive advantage by continuously improving the effectiveness and efficiency of the SE processes used by the organization.
Mange Product Line Evolution	Establish and provide the necessary resources for acquiring, developing, and applying technology to a product line for competitive advantage.
Manage SE Support Environment	Provide the technology environment needed to develop the product and perform the process.
Provide Skills and Knowledge	Provide the organization with the necessary skills to perform the work using all project needs and organizational goals as the basis for the needs.
Coordinate with Suppliers	Provide suppliers with clear expectations and measures of effectiveness. Communicate frequently with suppliers.

Tabela 7 - Áreas de Processo do SE-CMM (CUSICK,kerinia)

Para cada área de processo, estão associadas níveis de capacidade ou níveis de maturidade necessários para identificar como as empresas evoluem durante o ciclo de vida de desenvolvimento de sistemas. A Tabela 8 ilustra os níveis de maturidade e as características comuns em cada nível.

Maturity Level	Capability Level	Common Features
0	Not Performed	
1	Performed Informally	• Base practices performed
2	Planned and Tracked	• Committing to perform • Planning performance • Disciplined performance • Tracking performance • Verify performance
3	Well Defined	• Defining a standard process • Tailoring the standard process • Using data • Perform the defined process
4	Quantitatively Controlled	• Establishing measurable quality goals • Determining process capability to achieve goals • Objectively managing performance
5	Continuously Improving	• Establishing quantitative process effectiveness goals • Improving process effectiveness

Tabela 8 - Níveis de maturidade e características comuns (Hanna, M)

O objetivo dos níveis de maturidade é estruturar o modelo por ordem de dificuldade, incentivando as empresas a primeiramente aprender o básico da engenharia de sistemas antes de efetivamente implantar um padrão organizacional. Desta forma, empresas que desejam implantar um modelo de processo conseguem identificar mais facilmente por onde começar.

3.4.1 Diferenças Entre os Modelos SE-CMM, CMMI e SECAM

De acordo com o *Internacional Council on Systems Engineering, o* modelo *SECAM* é uma ferramenta utilizada para medir e avaliar a capacidade de engenharia de sistemas das organizações, para identificar as áreas com problemas e prover um vetor de crescimento da capacidade. Este modelo está estruturado por categorias de

processos contendo áreas chaves (*KPA – Key Focus Área).* No modelo *SE-CMM,* como também no *CMMI,* estas áreas são chamadas de “Áreas de Processos”.

Cada área chave contém níveis ascendentes de capacidade de engenharia de sistemas e estão divididas entre três categorias de processo (Gerenciamento, Organização e Engenharia de Sistemas). No modelo *CMMI* as “Áreas de Processos” são agrupadas em quatro categorias (Gerenciamento de Processos, Gerenciamento de Projetos, Engenharia, Suporte). A Tabela 9 apresenta as áreas chaves divididas por categorias de processo do modelo *SECAM*.

1.0 Management	2.0 Organization	3.0 Systems Engineering
1.1 Planning	2.1 Process Management and Improvement	3.1 System Concept Development
1.2 Tracking and Oversight	2.2 Competency Development	3.2 Requirements and Functional Analysis
1.3 Subcontract Management	2.3 Technology Management	3.3 System Design
1.4 Inter-group Coordination	2.4 Environment and Tool Support	3.4 Integrated Engineering Analysis
1.5 Configuration Management		3.5 System Integration
1.6 Quality Management		3.6 System Verification
1.7 Risk Management		3.7 System Validation
1.8 Data Management		

Tabela 9 - Áreas Chave por Categorias de Processo do modelo SECAM (INCOSE, 1996)

No modelo *SECAM* são considerados seis níveis de capacidade. Estes níveis são utilizados para descrever o nível de capacidade que as empresas estão realizando em suas atividades de engenharia de sistema. Em cada nível existe um conjunto de atributos que definem o que deve ser realizado e/ou não realizado para se atingir um determinado nível. Observa-se que os níveis aplicados no modelo *SECAM* são semelhantes aos níveis de maturidades aplicados no modelo *SE-CMM*, existindo diferenças apenas na quantidade de atributos e/ou características de cada

nível. A Tabela 10 apresenta o exemplo dos atributos de capacidade para a Engenharia de Sistemas.

Capability		Process Attributes	Non-Process Attributes
5	Optimizing	• program process effectiveness goals are established based upon business goals • continuous process improvement of program processes • continuous process improvement of standards	• activities driven by systems eng. benefit • fully scalable complexity management • SE focus is product life cycle & strategic applications • activities are optimally effective • work products are of optimal utility
4	Measured	• metrics derived from proc data • quantitative understanding of program processes • ability to predict performance • program process induced defects identified • program processes improved	• all information fully integrated in a program database • activities driven by systems eng. benefit • SE focus on all phases of product life cycle • activities are measurably effective • work products are of measurably significant utility
3	Defined	• processes are defined by org standards • standards are tailored & used • tailoring is reviewed & approved • program processes data is collected • customer feedback is obtained	• consistent program success • all information is managed electronically • key information is integrated in a program database • activities driven by benefit to program • SE focus is requirements through operation • activities are significantly effective • work products are of significant utility
2	Managed	• policies define need for activities • activities are planned, tracked & verif • work products reviewed for adequacy • corrective actions are taken • work products are controlled	• key information managed electronically • activities driven by benefit to customer • SE focus is requirements through design • activities are adequately effective • work products are of adequate utility
1	Performed	• activities done informally • non-rigorous plans & tracking • dependency on "heros" • work products are in evidence • general recognition of need for activity	• information is paper-based • activities driven only by contract • SE focus limited to requirements • activities are marginally effective • work products are of marginal utility
0	Initial	• general failure to perform activities • no easily identifiable work products • no proof something was accomplished	• no assurance of success • information is difficult to identify • driving force for activities is indeterminate • no assurance of complexity management • no systems engineering focus • activities and products of little effect or value

Tabela 10 - Atributos de capacidade para a Engenharia de Sistemas do modelo SECAM (INCOSE, 1996)

3.4.2 Integrando o *RUP-SE* com o *CMMI*

Uma organização atinge níveis de capacidade em Engenharia de Sistemas aplicando padrões para desenvolver processos de engenharia de sistemas, dispondo de tecnologia e pessoas com conhecimentos apropriados para especificar as necessidades e o domínio do produto.

O *RUP-SE* determina processos que definem as melhores práticas para a engenharia de sistemas. Ao implementar o *CMMI* integrado com o *RUP,* as organizações conseguem ser auxiliadas no controle e no monitoramento dos processos do *RUP* e na entrega de sistemas com garantia de qualidade.

Ao mapear o *RUP-SE* e o *CMMI* verifica-se os elementos que são comuns em cada processo e como ambos podem ser relacionados. A Tabela 11 apresenta esta relação.

CMMI Element	Maps to RUP element
Process Area	Workflow/Activity
Specific Practices/Goals	Activities/Tasks
Generic Practices/Goals	Activities/Tasks
Sub practices/goals	Activities/Tasks
Work products	Work products/Templates/Checklists/Guidelines

Tabela 11 - Relação de elementos CMMI x RUP (The Rational Edge, out. 2007)

De acordo com UTTANGI e RIZWAN (2007) é comum as empresas desejarem implementar o RUP e o CMMI definindo os processos de ambos separadamente. Isto pode criar redundância na documentação dos processos. Para solucionar este problema recomenda-se melhorar o processo *RUP* já implementado para torná-lo compatível com as boas práticas do framework *CMMI*, ao invés de implementá-los separadamente.

4 CONCLUSÕES

O desenvolvimento de software não é uma atividade simples. Para se construir um software com qualidade, processos devem ser seguidos com o intuito de organizar as atividades fundamentais a serem realizadas e a abordagem que será adotada na elaboração do software, pois diferentes tipos de sistemas necessitam de diferentes processos de desenvolvimento.

Quando pensamos em sistemas, além do desenvolvimento do software temos de considerar todo o contexto do sistema, ou seja, todos os elementos envolvidos onde o software é apenas um destes elementos.

A Engenharia de Sistemas disponibiliza meios para lidar com sistemas complexos e engloba todas as disciplinas relacionadas à Engenharia de Software, formando um processo de desenvolvimento estruturado que considera tanto as necessidades de negócio quanto as necessidades técnicas de todos os clientes, com o objetivo de prover a qualidade do produto de acordo com as necessidades dos usuários.

Compreender a Engenharia de Sistema permite a aplicação dos melhores processos ao se construir sistemas complexos, como por exemplo os sistemas de automação. Desta forma, é possível visualizar o sistema como um todo, dividí-lo em sub-sistemas e gerenciá-lo adequadamente garantindo o melhor atendimento das necessidades dos clientes (patrocinadores do projeto), assim como dos usuários.

De acordo com Sommerville (2007), o uso de um processo de software inadequado pode reduzir a qualidade ou a utilidade do produto de software a ser desenvolvido e/ou aumentar os custos de desenvolvimento.

Podemos considerar também esta observação realizada pelo Sommerville (2007) quando falamos de Engenharia de Sistemas. Lembrando que a definição de um processo dependerá do contexto do projeto e da organização.

Para suportar as necessidades da Engenharia de Sistemas, adaptações aos processos e às atividades realizadas durante o desenvolvimento de software foram realizadas, tais como a extensão do RUP para a Engenharia de Sistemas. Outras adaptações foram realizadas conforme foi apresentado neste trabalho.

4.1 IMPACTOS DA ENGENHARIA DE SISTEMAS

De acordo com estudos realizados por (SNYDER,1991) os esforços da engenharia de sistemas podem influenciar o desenvolvimento de softwares. Estes estudos apontam que podem ocorrer problemas em algumas áreas durante a transição de uma fase de desenvolvimento da engenharia de sistemas para uma fase da engenharia de software.

Geralmente o engenheiro de sistemas começa com a especificação dos requisitos funcionais do sistema, definindo o que o software deve fazer. O engenheiro de software tenta sintetizar estes requisitos criando o projeto do software. No entanto, existe grande dificuldade em produzir o projeto de software a

partir da especificação dos requisitos, pois a maneira como os requisitos são derivados, pelos engenheiros de sistemas, pode-se influenciar bastante no desenvolvimento dos softwares subseqüentes.

A fase de análise de sistemas permite a decomposição do sistema em subsistemas com o objetivo do entendimento total do sistema. A especificação final dos requisitos de software descreve o conjunto de requisitos que o projeto de software deve satisfazer. A metodologia de análise utilizada e o resultado da decomposição do software podem causar um impacto alto na liberdade do engenheiro de software durante a elaboração do projeto do software. E para minimizar estes impactos a engenharia de sistemas deve considerar os esforços sobre a engenharia de software nos seguintes momentos:

- Ao definir os hardwares e softwares que irão atender os requisitos dos usuários;
- Ao criar os documentos que especificam os níveis de detalhes do projeto. Sendo necessário eliminar o excesso de detalhes, mantendo no documento apenas o que for requerido pela aplicação.
- Ao controlar as rastreabilidades.

É necessário um maior empenho para que ambas as disciplinas (engenharia de software e engenharia de sistemas) possam caminhar juntas desempenhando as funções as quais lhe são atribuídas sem que o trabalho realizado em uma disciplina impacte o trabalho na outra.

4.2 ÁREAS DE COMPETÊNCIAS

Assim como a engenharia de software possui um guia desenvolvido pelo *IEEE Computer Society* para estabelecer uma base de conhecimento da engenharia de software, conhecido como *SWEBOK,* o *International Council On Systems Engineering* (*INCOSE)* também desenvolveu um guia contendo as áreas de competências relacionadas à engenharia de sistemas. Esta base de conhecimento é conhecida como *G2SEBoK* e procura definir a engenharia de sistemas em termos de conceitos, processos, competências e capacidades. A versão inicial deste guia contempla as áreas de conhecimento:

- Processos de Negócio e Avaliação Operacional
- Sistema / Solução / Teste de Arquitetura
- Custo do Ciclo de Vida / Análise de Custo-Benefício
- Manutenção / Logística
- Modelagem, Simulação e Análise
- Gerenciamento: Riscos, Configuração, Linha-Base

4.3 TRABALHO FUTURO

Como trabalho futuro, podem ser identificadas as seguintes atividades:

- Criação de um *case* para aplicação dos conceitos da engenharia de sistemas.
- Mapeamento das necessidades da engenharia de sistemas com os processos e/ou ferramentas existentes que oferecem suporte a esta área.

REFERÊNCIAS BIBLIOGRÁFICAS

PRESSMAN,R. *Engenharia de Software.* 6. ed. São Paulo: McGraw-Hill, 2006.

SOMMERVILLE,I. *Engenharia de Software.* 8.ed. São Paulo: Pearson Addison-Wesley, 2007.

NISE,N. *Engenharia de Sistemas de Controle*. 3.ed. Rio de Janeiro: LTC, 2002.

BOOCH,G.; RUMBAUGH,J.; JACOBSON,I. *UML Guia do Usuário.* Tradução Fábio Freitas da Silva e Cristina de Amorim Machado. Rio de Janeiro: Elsevier, 2005.

IEEE COMPUTER SOCIETY. Guide to the Software Engineering Body of Language. California: IEEE,2004

Naur,P. e Randall, B., (eds.), *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee ,*NATO, 1969.

CANTOR,Murray; ROOSE,Gene. Hardware/Software codevelopment using a model-driven systems development (MDSD) approach. **The Rational Edge,** dez. 2005. Disponível em:< www.ibm.com/developerworks/rational/library/dec05/cantor/index.html>. Acesso em: 01 nov. 2009.

HAUMER,Peter. IBM Rational Method Composer: Part 1: Key concepts. Disponível em: < www.ibm.com/developerworks/rational/library/dec05/haumer>. Acesso em: 17 set. 2009.

Rational Unified Process. Disponível em: < http://www.wthreex.com/rup/portugues/index.htm>. Acesso em: 27 jul. 2009.

OMG Systems Modeling Language. Disponível em: <http://www.omgsysml.org>. Acesso em: 22 set. 2009.

Wikipedia – Systems Modeling Language. Disponível em: <http://en.wikipedia.org/wiki/SysML >. Acesso em: 22 set. 2009.

SysML Fórum. Disponível em: < http://www.sysmlforum.com/FAQ.htm>. Acesso em: 23 set. 2009.

Guide to Systems Engineering Body of Knowledge (G2SEBoK). Disponível em: < http://www.incose.org/practice/guidetosebodyofknow.aspx> . Acesso em: 25 out. 2009.

OMG Object Management Group. Disponível em: <http://www.omg.org>. Acesso em 30 out. 2009.

CUSICK, Kerinia. The Systems Engineering Capability Maturity Model: Where to start?. IEEE Aerospace & Electronic Systems Magazine.

INCOSE. Systems Engineering Capability Assessment Model. Jun. 1996. Disponível em: < http://www.incose.org/ProductsPubs/pdf/SECAM-SysEngCapabilityAssessModel_1996-06.pdf>. Acesso em: 02 nov. 2009.

DEFENSE ACQUISITION UNIVERSITY PRESS. Department of Defense (DoD). **Systems Engineering Fundamentals**. Virginia, 2001. Disponível em: < http://www.dau.mil/pubscats/Pages/sys_eng_fund.aspx>. Acesso em: 01 nov. 2009.

RUP ® SE1.1. A Rational Software White Paper. **Rational Unified Process for Systems Engineering**.

BOCK,Conrad. Systems engineering in the product lifecycle. 2005. Disponível em: < http://www.mel.nist.gov/msidlibrary/doc/sysmlplm.pdf> .Acesso em: 17 nov. 2009.

HANNA, M. Systems Engineering Maturity and Benchmarking. **Software Productivity Consortium**. Set. 1996 .

UTTANGI, Roshan; RIZWAN, Syed. Fast track to CMMI implementation: Integrating the CMMI and RUP process framework. **The Rational Edge,** out. 2007. Disponível em:< www.ibm.com/developerworks/rational/library>. Acesso em: 01 nov. 2009.

SNYDER, Charles. System Engineering impact on software development. 1991. ACM 0-89791-445-7/91/1000-0425.