PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

CURSO DE ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

RAFAEL AUGUSTO DE BARROS

ESPECIFICAÇÃO DE REQUISITOS BASEADO NO MÉTODO COSMOD-RE

São Paulo/SP

2016

Rafael Augusto de Barros

Especificação de requisitos baseado no método COSMOD-RE

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. Ms. Carlo Borsoi Moura.

São Paulo/SP

2016

AGRADECIMENTOS

A todos familiares por demonstrarem seu apoio, incentivo e confiança a completar este trabalho.

Ao meu orientador Prof. Ms. Carlo Borsoi Moura, pela oportunidade.

Qualidade sem quantidade total não é qualidade, é privilégio.

(autor desconhecido)

RESUMO

Um dos maiores problemas encontrados na Engenharia de Requisitos é o fato dos requisitos estarem mal especificados e inconsistentes com a necessidade do cliente.

Esta monografia propõe um modelo de processo visando auxiliar a fase de especificação de requisitos, de forma que seja possível concentrar tempo na análise dos requisitos, reduzindo incertezas e aumentando a qualidade. A proposta é realizada através da aplicação do método COSMOD-RE, que é capaz de apoiar o desenvolvimento interligado de requisitos e artefatos de arquitetura para sistemas de software.

O método proposto é composto de 3 fases e atuará na área de Análise, Especificação e Validação de Requisitos auxiliando o Engenheiro de Requisitos.

Por fim, foi apresentado um estudo experimental e comparativo entre o método proposto e um estudo de caso para avaliar e validar o processo proposto, identificando benefícios da sua utilização no desenvolvimento de software. Destacando-se como pontos positivos do estudo experimental o tempo total de envolvimento, considerado aceitável pelos envolvidos e a importância da identificação dos cenários de testes, controle e planejamento.

Palavras Chaves: Processo de Engenharia de Requisitos; Especificação e Validação de Requisitos; Qualidade na Descrição de Requisitos; COSMOD-RE;

ABSTRACT

REQUIREMENT SPECIFICATION BASED IN THE COSMOD-RE METHOD

One of the major problems encountered in Requirements Engineering is that the requirements are poorly specified and inconsistent with the customer's needs.

This monograph proposes a process model to assist the requirement specification phase, so that it is possible to concentrate time on the requirements analysis, reducing uncertainties and increasing quality. The proposal is carried out through the application of the COSMOD-RE method, which is able to support the interconnected development of requirements and architecture artifacts for software systems.

The proposed method is composed of 3 phases and will act in the area of Analysis, Specification and Validation of Requirements assisting the Requirements Engineer.

Finally, an experimental and comparative study was presented between the proposed method and a case study to evaluate and validate the proposed process, identifying benefits of its use in software development. Emphasizing the positive points of the experimental study were the total time of involvement, considered acceptable by those involved and the importance of identifying the test, control and planning scenarios.

Keywords: Requirements Engineering Process; Specification and Validation of Requirements; Quality in the Description of Requirements; COSMOD-RE;

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ES	Engenharia de *Software*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ISO	*International Organization of Standardization*
RUP	*Rational Unified Process*
SRS	*Software Requirements Specification*
UML	*Unified Modeling Language*

SUMÁRIO

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

A entrega de software com qualidade, dentro dos prazos estabelecidos, custos esperados e atendendo as necessidades dos envolvidos é um desafio atualmente vivenciado por todos da indústria de software.

Essas mesmas indústrias de software estão cada vez mais interessadas em garantir que o software desenvolvido represente, realmente, uma solução para o qual foi proposto, ou seja, um produto final que represente as necessidades dos *stakeholders*[1].

Uma pesquisa americana realizada pelo Instituto Standish Group, utilizando uma amostra de 365 companhias entrevistadas, representando 8.380 aplicações, chegou às seguintes informações:

Os EUA gastam US$ 250 bilhões por ano em desenvolvimento de aplicação de tecnologia da informação em aproximadamente 175 mil projetos.

31,1% de todos os projetos foram cancelados antes do seu término, representando um desperdício da ordem de US$ 81 bilhões.

52,7% de todos os projetos chegaram ao final tendo custado 189% do valor estimado, representando US$ 59 bilhões em custo adicional, atrasaram em até 222% da estimativa original além de serem entregues com apenas 61% das características originalmente especificadas.

16% foram entregues no prazo e dentro do orçamento.

Frente aos principais fatores de criticidade de um projeto como mostrado no gráfico abaixo, podemos identificar a necessidade de melhorias nas formas de entendimento, elaboração e gerenciamento dos requisitos de um sistema. Os

[1] Indivíduo ou organização que tem um direito, ação, declaração ou interesse em um sistema ou na posse das características do sistema que satisfaçam suas necessidades e expectativas [ISO12207].

problemas relacionados com os requisitos são os primeiros da lista e totalizam quase 40% das causas de dificuldade de um projeto.

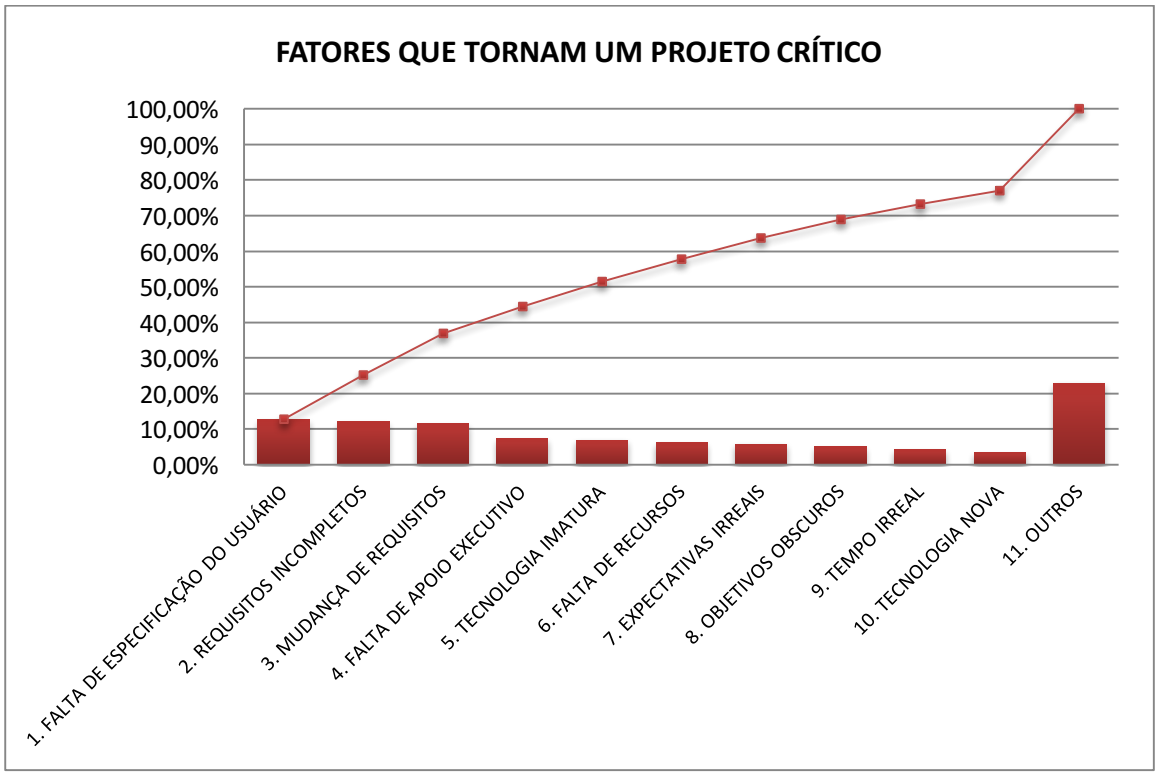

Figura 1 - Principais fatores que tornam um projeto crítico.

De acordo com Pressman [PRES2011], o custo de correção de defeitos na fase de projeto é cerca de três a seis vezes mais alto do que na fase de definição de requisitos, e este custo aumenta ainda mais quando a correção de defeitos é realizada em fases mais avançadas do processo de desenvolvimento, como na fase de teste.

Portanto, por que deixar a busca e eliminação de defeitos para as últimas fases do processo de desenvolvimento sabendo-se que quanto mais se distancia das fases iniciais mais caro ficará sua posterior correção? Como identificar os defeitos antes do início da construção do produto?

Dentre uma vasta lista de problemas, alguns dos mais recorrentes seriam, por exemplo:

Requisitos não utilizados: alguns requisitos são desenvolvidos, porém não são utilizados após a implantação do sistema. Será que realmente estes requisitos deveriam ter sido implementados?

Requisitos desenvolvidos em desacordo com o que foi solicitado: funcionalidades desenvolvidas, porém quando apresentadas ao *stakeholder* estão em desacordo com o que foi solicitado. Será que todos os envolvidos entenderam qual era a expectativa e realmente desenvolveram o que foi solicitado?

Busca da perfeição e completude na versão 0: muitas equipes de desenvolvimento planejam a entrega total do sistema em uma única versão. Será que a priorização e a entrega de requisitos essenciais não ajudariam a validar o que realmente é necessário para a conclusão do sistema em uma versão n?

Ter um processo definido significa ter maior possibilidade para se obter, entender e validar as necessidades e expectativas do cliente para, posteriormente, documentá-las visando garantir a qualidade e aumento da satisfação das partes interessadas, viabilizando, por exemplo, a criação de documentos de especificação de requisitos (SRS), modelos de casos de uso, entre outros artefatos[2], que resultarão na conclusão, com êxito, de um acordo entre quem solicita e quem desenvolve, estabelecendo clara e rigorosamente o que deverá ser produzido.

De acordo com artigos, literaturas e pesquisas relacionadas a desenvolvimento de software, grande parte dos engenheiros de software utilizam as metodologias disponíveis, mas não focam no objetivo das mesmas ou desconhecem o processo por elas apoiados. Também existem os que são subordinados a gerências que não dão o devido valor. O fato é que se a especificação do software é subjetiva, o resultado do trabalho provavelmente será um software com nível de qualidade abaixo das expectativas.

[2] Artefato é um pedaço de informação que é produzida, modificada ou usada por um processo, define uma área de responsabilidade e está sujeita a controle de versão. Um artefato pode ser um modelo, um elemento de modelo ou um documento [KRUCHTEN2003]. Um artefato é o termo usado para qualquer produto de trabalho: código, gráficos para web, esquemas de banco de dados, documentos de texto, diagramas, modelos e assim por diante [LARMAN2007].

Com intuito de definir uma hierarquia em camadas de abstração, visando a proximidade na definição dos requisitos e artefatos de arquitetura, Klaus Pohl [POHLKLAUS] propôs o método COSMOD-RE que é estruturado em três blocos de construção e apoia o desenvolvimento simultâneo de requisitos e artefatos de arquitetura relacionados.

Pohl considera quatro requisitos essenciais para o método, desenvolvimento em paralelo de requisitos e artefatos de arquitetura, suportar o alinhamento entre os requisitos e artefatos de arquitetura, apoiar a definição de requisitos (em pormenor) com base em artefatos de arquitetura e fornecer camadas de abstração claramente definidas.

O COSMOD-RE foi descrito no guia como um método voltado para sistemas intensivos embarcados.

Esta pesquisa visa realizar uma avaliação deste método realizado de uma maneira diferente da qual foi idealizada e exemplificada no guia por Pohl.

Como o método não foi utilizado em aplicações de software de gestão administrativa (não embarcado), foi identificada uma oportunidade para realização da pesquisa.

A presente pesquisa será realizada em uma grande organização cuja o processo de engenharia de requisitos atualmente não está bem definido, contribuindo para ampliar a aplicação do método proposto por Pohl.

1.2. OBJETIVO

O objetivo específico desta pesquisa é avaliar se o método COSMOD-RE é adequado para produção de uma especificação de requisitos que influencie diretamente o processo de desenvolvimento de software de gestão administrativa, de forma que a qualidade dos requisitos seja comprovadamente atestada.

1.3. RESULTADOS ESPERADOS

Como resultado deste trabalho é esperado identificar e definir um processo para a especificação de requisitos através do método COSMOD-RE no desenvolvimento de produtos de software, em ambientes de desenvolvimento baseado em clientes internos (a organização é fornecedora e cliente ao mesmo tempo).

Além de definido e analisado em função de outros modelos e processos já existentes na literatura de Engenharia de Requisitos, busca-se também verificar e validar a ocorrência de benefícios através da aplicação do método COSMOD-RE no desenvolvimento de software. Um estudo experimental será realizado para avaliar a viabilidade, melhoria e produtividade do modelo de processo proposto.

Algumas outras abordagens propostas destacam-se:

- Facilitar a manutenção da documentação, das características e dos modelos, por incorporação das variações requeridas e documentadas nas várias implementações.

- Facilitar a comunicação das especificações e modificações dos requisitos até a customização, entre analistas, implementadores e clientes.

- Criação de uma matriz para mapeamento e rastreabilidade de requisitos.

1.4. MÉTODO DE PESQUISA

A pesquisa propõe-se a avaliar um método chamado COSMOD-RE com base numa pesquisa e aplicada em uma empresa, voltado diretamente ao processo de desenvolvimento de um sistema de gestão administrativa.

Inicialmente foi feita uma pesquisa bibliográfica, com a finalidade de compreender o estudo de obras dos principais autores que desenvolveram trabalhos relacionados com a identificação, análise e negociação, especificação e documentação e validação de requisitos que objetiva identificar as

metodologias, processos e ferramentas conceituadas na literatura relacionada à Engenharia de Requisitos e estudo do método COSMOD-RE.

A Pesquisa Bibliográfica se concentra nos seguintes tópicos:

- Descoberta, descrição, análise e representação de requisitos no ciclo de desenvolvimento de software e os principais fatores críticos que influenciam diretamente a probabilidade de sucesso de um projeto de software.

- Conceitos e taxonomia utilizados para a rastreabilidade.

- Definição e conceituação do método COSMOD-RE e seu guia explicativo.

Caracterização de sistemas de software de Gestão Administrativa (ERP) que fornecem serviços através de módulos, com identificação de elementos presentes nas arquiteturas destes sistemas, e possíveis fatores de risco que podem comprometer a aceitação dos mesmos.

Aplicações do processo proposto em projetos de software. Nessa etapa, foi selecionado e apresentado um projeto de software significativo que o autor vivenciou durante sua vida profissional. O método COSMOD-RE foi aplicado a tal projeto de software e seus resultados finais serão analisados.

1.5. ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 6 capítulos, resumidos a seguir:

O **capítulo I** apresenta à proposta do trabalho, o contexto do problema, a motivação do estudo, os objetivos a serem alcançados e sua estrutura de pesquisa.

O **capítulo II** trata da fundamentação teórica por meio de assuntos envolvidos na proposta: desde a concepção até a entrega de software com qualidade. São relacionados trabalhos sobre Engenharia de Software, Engenharia de Requisitos,

RUP, Metodologias Ágeis, com definições de conceitos, metodologias, processos, técnicas, características e ferramentas.

O **capítulo III** descreve o método COSMOD-RE e todos os blocos que fazem parte do seu guia.

O **capítulo IV** concentra-se na descrição dos conceitos, técnicas e práticas relacionadas ao processo proposto como a identificação/levantamento, a análise e negociação, a especificação e documentação, e validação de requisitos. Neste capítulo também são apresentadas possíveis maneiras de se avaliar a eficiência do processo proposto considerando a minimização dos riscos, aumento da qualidade e da produtividade.

O **capítulo V** descreve os resultados da aplicação do processo em um projeto de software vivenciado pelo autor e as principais dificuldades observadas durante a sua utilização.

O **capítulo VI** apresenta um conjunto de resultados obtidos pela pesquisa, destacando a aplicação do método COSMOD-RE e a sua influência no processo definido de desenvolvimento de software como principal instrumento na especificação de requisitos. Além disso, conclusões diversas e sugestões para possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. ENGENHARIA DE SOFTWARE

De acordo com a IEEE, a Engenharia de Software (ES) é a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software.

Partindo dessa definição, dia após dia surgem novas técnicas, métodos e ferramentas que auxiliam os profissionais de TI a executarem melhor as atividades envolvidas no desenvolvimento de software, todas sobre o pilar da qualidade. Nas próximas seções são mostrados alguns temas relevantes a esta pesquisa e associados a Engenharia de Software.

2.1.1. FUNDAMENTOS DA ENGENHARIA DE SOFTWARE

[PRESS2011] descreve ES como uma tecnologia em camadas e como qualquer abordagem em engenharia, deve estar fundamentada em um compromisso organizacional com a qualidade. As camadas compreendem processos, métodos e ferramentas para o desenvolvimento de software.

A base para a ES é a camada de processos. O processo de ES é a ligação que mantém unidas as camadas de tecnologia e permite o desenvolvimento de software racional e dentro do prazo. O processo forma a base para o controle do gerenciamento de projetos de software, estabelece o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são geridas de forma apropriada.

Os métodos fornecem a técnica de como fazer para construir o software. Os métodos incluem um amplo conjunto de tarefas que abrange análise de requisitos, projeto, construção, manutenção e testes. Métodos de ES baseiam-se em um conjunto de princípios básicos, que governam cada área

de tecnologia e incluem atividades de modelagem e outras técnicas descritivas.

As ferramentas fornecem suporte automatizado ou semiautomatizado para os processos e métodos.

2.1.2. PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Conforme citação de Ivar Jacobson, Grady Booch e James Rumbaugh, “Um processo define quem está fazendo o quê, quando e como para atingir determinado objetivo”.

Na definição de [PRES2011], processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto de trabalho. A atividade esforça-se para cumprir um objetivo maior e é utilizada independentemente do campo de aplicação, do tamanho do projeto, da complexidade de esforços. A ação envolve um conjunto de tarefas que resultarão em um artefato de software fundamental (por exemplo, um modelo de arquitetura). Por fim a tarefa concentra-se no objetivo menor/pequeno, porém, bem definido e produz um resultado tangível.

O processo de desenvolvimento de software incorpora cinco atividades estruturais genéricas que se aplicam a todos os projetos de software, são elas:

Comunicação: a intenção é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.

Planejamento: criar um "mapa" ajuda a guiar a equipe na sua jornada, é definido o trabalho de ES, descrevendo as tarefas técnicas a ser conduzida, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a ser produzidos e um cronograma de trabalho.

Modelagem: o objetivo é criar um "esboço" do que será feito, de modo que se possa ter uma ideia do todo. Se necessário, refina-se o esboço com mais detalhes, numa tentativa de compreender melhor o problema e como resolvê-lo.

Construção: combinação de geração de código (manual ou automatizado) e testes necessários para revelar erros na codificação.

Emprego: entrega do software para o cliente, que avalia o produto entregue e fornece feedback, baseado na avaliação.

2.1.2.1. MODELOS DE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Os modelos de ciclo de vida do software podem ocorrer em sequência ou sobrepostas e executadas interativamente, dependendo do modelo adotado para a construção do software. Podem ser decompostas em unidades menores, como atividades, ações ou tarefas, a fim de permitir um maior controle e visibilidade da execução do projeto de software.

Os modelos de ciclo de vida de software mais comumente citados na literatura são:

Cascata: é um modelo de desenvolvimento bem definido, no qual os processos são executados de forma sequencial. PRESSMAN [PRES2011] cita que algumas vezes também são chamados de clico de vida clássico, e sugere uma abordagem sequencial e sistemática. Os principais estágios dos modelos cascata retratam as atividades de desenvolvimento fundamentais, que são: Comunicação (início do projeto, levantamento das necessidades), Planejamento (estimativas, cronogramas, acompanhamento), Modelagem (análise, projeto), Construção (codificação, testes) e Emprego (entrega, suporte, feedback).

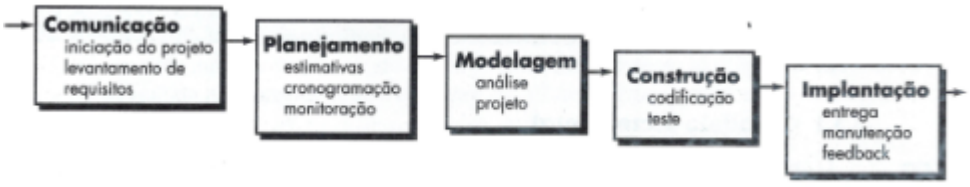

Figura 2 - Modelo Cascata [PRES2011]

Incremental: o modelo incremental combina fluxos lineares e paralelos, de forma escalonada, à medida que o tempo avança. Cada sequência linear gera “incrementais” do software. Esse modelo tem seu foco voltado para a entrega de um software operacional a cada incremento.

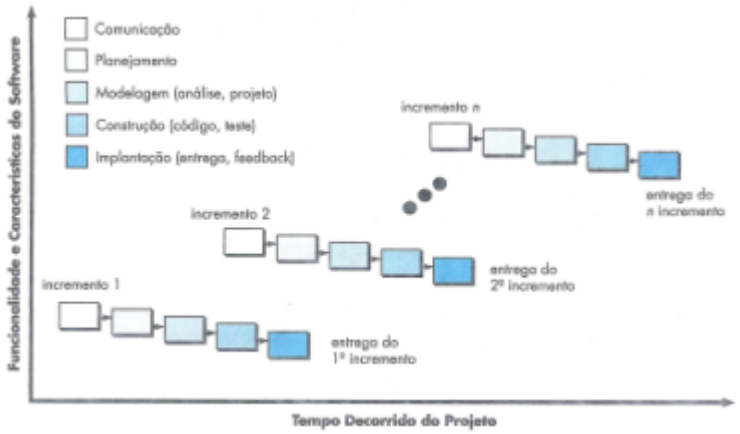

Figura 3 - Modelo incremental [PRES2011]

2.1.3. RUP – PROCESSO UNIFICADO

Para [KRUCHTEN2003], o RUP - Rational Unified Process, embora sugira um processo, pode ser considerado como:

Uma abordagem de desenvolvimento de software que é iterativa, centrada na arquitetura e dirigida por casos de uso, ou seja, levantamento de requisitos baseados na visão do usuário.

De certa forma, o Processo Unificado é uma tentativa de aproveitar os melhores recursos e características dos modelos tradicionais de processo de software, mas caracterizando-os de modo a implementar muitos dos melhores princípios do desenvolvimento ágil de software.

A figura abaixo ilustra como o RUP está organizado e estruturado em fases e disciplinas.

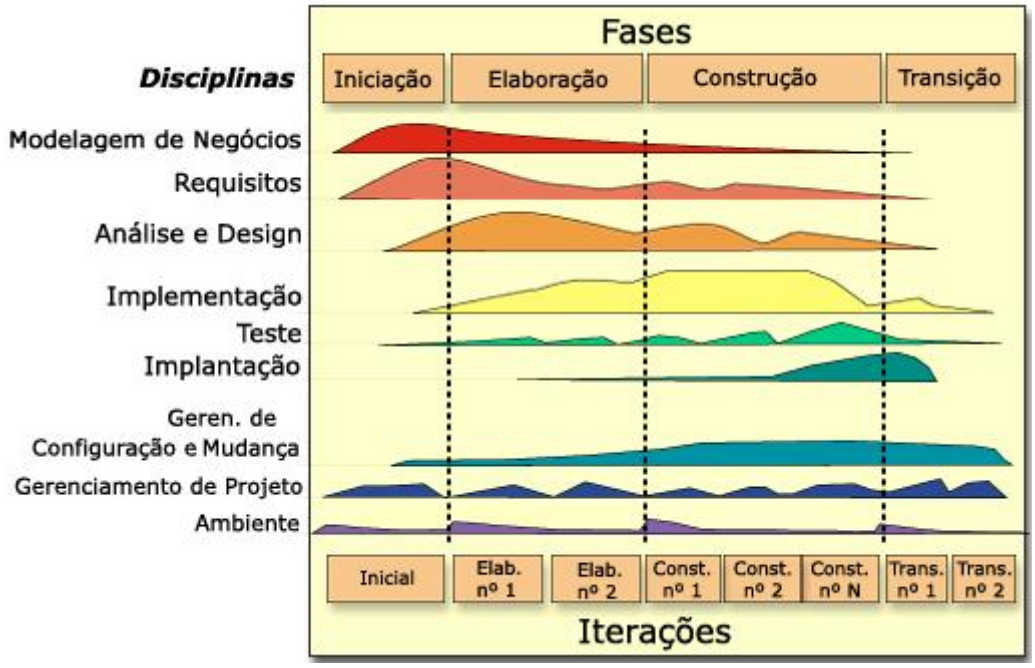

Figura 4 - Visão Geral - RUP - Rational Unified Process - [KRUCHTEN2003]

O processo de desenvolvimento de software do RUP é iterativo, no qual uma iteração incorpora um conjunto de atividades em modelagem de negócios, requisitos, análise e projeto, implementação, testes e implantação, em várias proporções, dependendo de onde a iteração esteja localizada no ciclo de desenvolvimento.

Os principais benefícios da abordagem iterativa são a identificação e o tratamento dos principais riscos do projeto em tempo hábil.

Abaixo um resumo de cada fase:

Iniciação

- Estabelecer o escopo do projeto.
- Selecionar os casos de uso críticos do negócio.
- Buscar ao menos uma arquitetura candidata.
- Elaborar cronograma e estimativas de custos do projeto.
- Estimar os riscos do projeto.

- Preparar o ambiente de apoio ao projeto.

Elaboração

- Assegurar que a arquitetura, os requisitos e os planos estejam estáveis para prover o resto do projeto.
- Resolver os riscos de arquitetura do projeto.
- Produzir um protótipo.
- Demonstrar que a arquitetura suportará os requisitos.
- Estabelecer um ambiente de apoio para o projeto.

Construção

- Minimizar os custos de desenvolvimento.
- Atingir qualidade de forma rápida e prática.
- Obter versões utilizáveis o quanto antes.
- Completar análise, design e implementação.
- Assegurar-se de que o usuário está pronto para a entrega.
- Buscar paralelismo nas atividades de desenvolvimento.

Transição

- Realizar beta teste para validar o produto.
- Executar teste paralelo com sistemas legados.
- Converter base de dados existentes.
- Treinar usuários.
- Resolver problemas de implantação.

O RUP usa um padrão visual apoiado na Unified Modeling Language (UML) e possui três elementos-chave: casos de uso, arquitetura e desenvolvimento iterativo e incremental.

2.1.4. METODOLOGIAS ÁGEIS

Segundo [LARMAN2007] métodos de desenvolvimento ágil usualmente aplicam desenvolvimento iterativo e evolutivo de tempo limitado, empregam

planejamento adaptativo, promovem entrega, incrementam e incluem valores e práticas que encorajam agilidade, reposta rápida e flexível à modificação.

O termo “Metodologias Ágeis” tornou-se popular, em 2001, quando Kent Beck e outros especialistas da área de desenvolvimento de software assinaram o Manifesto Ágil, de forma que foi criada a Aliança Ágil baseada nos conceitos-chave abaixo:

- Indivíduos e interações acima de processos e ferramentas.
- Software operacional acima de documentação completa.
- Colaboração do cliente acima de negociação de contratos.
- Respostas as mudanças acima de seguir um plano.

O “Manifesto Ágil” não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparados com os indivíduos e interações, com o software estar operacional, com a colaboração do cliente e as respostas rápidas a mudanças e alterações. Estes conceitos aproximam-se melhor com a forma que pequenas e médias organizações trabalham e respondem a mudanças.

Dentre os modelos de processos ágeis, o mais amplamente utilizado de todos é o da Extreme Programming (XP) desenvolvida pelo Kent Beck. Porém, muitos outros têm sido propostos e encontram-se em uso no setor. Os mais comuns são:

- Desenvolvimento de software adaptativo.
- Scrum.
- Método de desenvolvimento de sistemas dinâmicos.
- Crystal.
- Desenvolvimento dirigido a funcionalidades.
- Desenvolvimento de software enxuto.
- Modelagem Ágil.
- Processo Unificado Ágil.

2.2. ENGENHARIA DE REQUISITOS

Para fundamentar as atividades da Engenharia de Requisitos, torna-se de certa forma obrigatória a compreensão do que são e como podem ser classificados os requisitos de um sistema.

Nesta seção, serão apresentadas as principais definições de requisitos e suas classificações, assim como alguns conceitos da Engenharia de Requisitos.

2.2.1. FUNDAMENTOS DA ENGENHARIA DE REQUISITOS

A Engenharia de Requisitos, pode ser descrita como “o ramo da Engenharia de Software que se preocupa com os objetivos, funções e restrições dos sistemas de software” [MACH2016].

De acordo com Pressman [PRES2011], a Engenharia de Requisitos constrói uma ponte para o projeto e para a construção. Fornece mecanismo apropriado para entender aquilo que o cliente deseja, analisando as necessidades, avaliando a viabilidade, negociando uma solução razoável, especificando a solução sem ambiguidades, validando a especificação e gerenciando as necessidades à medida que são transformadas em um sistema operacional.

Ela abrange sete tarefas distintas: concepção, levantamento, elaboração, negociação, especificação, validação e gestão. É importante ressaltar que algumas dessas tarefas ocorrem em paralelo e todas são adaptadas às necessidades do projeto.

Concepção: a maioria dos projetos começa quando é identificada a necessidade do negócio ou é descoberto um novo serviço ou mercado potencial. É definido um plano de negócio para a ideia e realizam uma análise de viabilidade descrevendo o escopo inicial do projeto.

Levantamento: questões sobre os objetivos do sistema e/ou produto são levantadas. O que deve ser alcançado, como deve atender as necessidades e

como deve ser utilizado no dia a dia. Aparentemente simples, porém nessa fase são identificados os maiores problemas relacionados ao desenvolvimento de software.

Elaboração: refinamento das informações obtidas durante a concepção e levantamento. Concentra-se no desenvolvimento de um modelo de requisitos refinado, identificando os diversos aspectos da função, do comportamento e das informações do software.

Negociação: consiste na conciliação dos conflitos dos pedidos "impossíveis" de serem alcançados, ou que divergem de outros requisitos levantados por outras partes interessadas.

Especificação: pode ser um documento escrito, ou um conjunto de modelo gráficos, um conjunto de cenários de uso. A sugestão é a criação de um "modelo-padrão" para utilização na especificação. Dessa forma os requisitos apresentados se tornam consistentes e, portanto, mais compreensível.

Validação: os artefatos produzidos são avaliados quanto à qualidade durante essa etapa. É examinado a especificação para garantir que todos os requisitos de software tenham sido declarados de forma não ambígua, que as inconsistências, omissões e erros tenham sido detectados e corrigidos, e por fim, que os artefatos estejam de acordo com os padrões estabelecidos para o processo, projeto e produto. O principal mecanismo é através de uma revisão técnica.

Gestão: os requisitos mudam, e isso persiste durante todo o ciclo de vida do sistema. A gestão de requisitos é um conjunto de atividades que ajuda a equipe de projeto a identificar, controlar e acompanhar as necessidades e suas mudanças a qualquer momento enquanto o projeto prossegue.

2.2.2. OS REQUISITOS DE SOFTWARE

[MACH2016] definiu requisito como sendo o ponto de partida para toda a definição do sistema e, consequentemente, são fatores decisivos no desenvolvimento do produto final.

A definição padronizada pela IEEE é a seguinte: (1) uma condição ou capacidade necessária a um usuário para resolver um problema ou alcançar um objetivo. (2) uma condição ou capacidade que deve ser alcançada ou possuída por um sistema ou por um componente de sistema para satisfazer um contrato, padrão, especificação ou outros documentos formalmente expostos. (3) uma representação documentada de uma condição ou capacidade como a dos itens 1 e 2.

Além de tratar um grande volume de informações que provém das mais diversas fontes, os projetistas de um software devem, ainda, estar atentos aos requisitos implícitos que fazem parte do produto, mas que não foram declarados por nenhum *stakeholder*.

2.2.3. A CLASSIFICAÇÃO DOS REQUISITOS

Segundo [HELIO2010], os requisitos podem ser divididos em:

- **Funcionais**, que são aqueles que definem funcionalidades ou ações que o sistema deve fornecer, geralmente podendo ser visualizados pelos casos de uso do sistema. Essas funcionalidades podem ser disponibilizadas a usuários e a outros sistemas.

- **Não funcionais**, descrevem atributos do sistema ou do ambiente do sistema, como por exemplo, usabilidade (considerando a facilidade de uso pelos usuários), confiabilidade (considerando o quão confiável é a utilização do sistema e a confiabilidade das informações apresentadas), entre outros.

Ainda seguindo a definições de requisitos, podemos também elencar os tipos de requisitos. De acordo com [HELIO2010] existe um padrão de categorização de requisitos pelo modelo conhecido como FURPS+, em que as letras da sigla significam *Functionality* (Funcionalidade), *Usability* (Usabilidade), *Reliability* (Confiabilidade), *Performance* (Desempenho) e *Supportability* (Suportabilidade).

O sinal + em FURPS+ significa requisitos que são igualmente importantes, tais como restrições de projeto que especificam ou restringem o projeto de um sistema, requisitos de implementação que especificam ou restringem padrões, linguagens, ambientes e outros para a construção do sistema, requisitos de interface que especificam itens externos com os quais o sistema deverá interagir, requisitos físicos que especificam ou restringem plataformas físicas, tais com hardware e rede.

Além das divisões em funcionais e não funcionais, e das características FURPS+, os requisitos também possuem atributos.

Com a finalidade de gerenciar requisitos, precisamos definir atributos para eles. Cabe ao gerente de projetos e ao engenheiro de requisitos definir quais serão os atributos a serem utilizados no projeto.

Conforme as boas práticas apresentadas por [HELIO2010], deve-se utilizar pelo menos os seguintes atributos para um projeto: benefício, estabilidade, situação, risco e responsável, além de um campo para observação. Abaixo, segue um resumo de cada atributo:

- **Benefício**: indica o grau de benefício do requisito relacionado às expectativas dos fornecedores de requisitos. Na vida real você poderá encontrar situações nas quais alguns fornecedores de requisitos declaram importância elevada a certos requisitos em detrimento a outros que consideram até desnecessários. Este atributo pode conter os seguintes valores:

- **Crítico**: quando o fracasso na implementação do requisito significar que o sistema não atenderá às expectativas dos interessados, tornando-se imprescindível para o sucesso do projeto.
- **Importante**: quando o não atendimento do requisito não determinar o fracasso, porém impacte na satisfação do usuário final da aplicação.

- **Desejável**: quando não impacta a satisfação do usuário final, podendo deixar de ser atendido.

- **Estabilidade**: esse atributo está associado diretamente a possibilidade de ocorrência de mudanças no requisito, indicando o grau de entendimento. Em situações em que o entendimento é superficial entre os próprios fornecedores de requisitos. Essa atribuo está associado à possibilidade de mudanças, portanto o gerenciamento de mudanças deve ser utilizado. Os valores seriam:

  - **Alta**: indica requisito com alto grau de entendimento e baixa probabilidade de mudança.

  - **Média**: indica requisito com considerável probabilidade de mudanças por ainda não estarem bem entendido pela equipe de projeto, ou pendências de definições dos fornecedores.

  - **Baixa**: representa alta probabilidade de mudança, devido à baixa maturidade.

- **Situação**: atributo que indica a situação do requisito, podendo conter os seguintes valores:

  - **Proposto**: relacionado a requisitos que ainda se encontram em análise pela equipe de projeto e até pelo próprio cliente. Esses requisitos não devem ser utilizados enquanto não passarem para o status de aprovado.

- **Aprovado**: aprovado pela equipe de projeto e fornecedores.

- **Cancelado**: foi desconsiderado do projeto e deve ser eliminado do escopo.

- **Risco**: deve ser gerenciado pela disciplina de gerenciamento de riscos do projeto. Em uma situação real, por exemplo, imaginemos o risco associados ao caso de uso de transferência de dinheiro entre contas de instituições financeiras. Esse requisito seria de alto risco. Os valores para esse atributo seriam:

  - **Alto**: requisito de alto risco por ter baixa estabilidade, com alta complexidade, contendo dependências externas.

  - **Média**: requisito de risco médio por ter estabilidade considerada média, com complexidade média, sem dependências externas.

  - **Baixo**: por ter alta estabilidade, baixa complexidade e sem dependências externas.

E por fim, é de suma importância termos um campo **Observação** para registro de pendências ou problemas relacionados, e um outro campo indicando o **Responsável** pelo requisito.

2.2.4. OS PROCESSOS DE ENGENHARIA DE REQUISITOS

A Engenharia de Requisitos envolve todas as atividades exigidas para criar e manter o documento de requisitos do sistema. Existem quatro atividades genéricas de processo de Engenharia de Requisitos que são de alto nível: o estudo do sistema, a obtenção e a análise de requisitos, a especificação de requisitos e sua documentação e, finalmente, a validação desses requisitos.

- **Estudo do sistema**: nesta fase é determinado o que o sistema deve fazer, definido o escopo do sistema. É o "descobrimento" do requisito.

Envolve 4 aspectos: domínio da aplicação, problema a ser solucionado, contexto de negócios e necessidades e restrições dos *stakeholder.*

- **Obtenção e Análise dos requisitos**: nesta fase os requisitos são "traduzidos" para os da engenharia de requisitos. Eles são entendidos e começam a ser transformados em especificações técnicas. É aqui que realmente começam a surgir as dúvidas.

- **Especificação dos requisitos**: nesta fase os requisitos são detalhados ainda mais e as eventuais dúvidas são sanadas.

- **Validação dos requisitos**: nesta fase os requisitos são finalmente aprovados pelos clientes e eventuais "recusas" fazem com que o requisito retorne ao estágio de descoberta/análise do requisito.

Não existe um único processo de Engenharia de Requisitos que atenda perfeitamente à todas as organizações. Entretanto a figura abaixo dá uma visão ampla dos principais processos existentes.

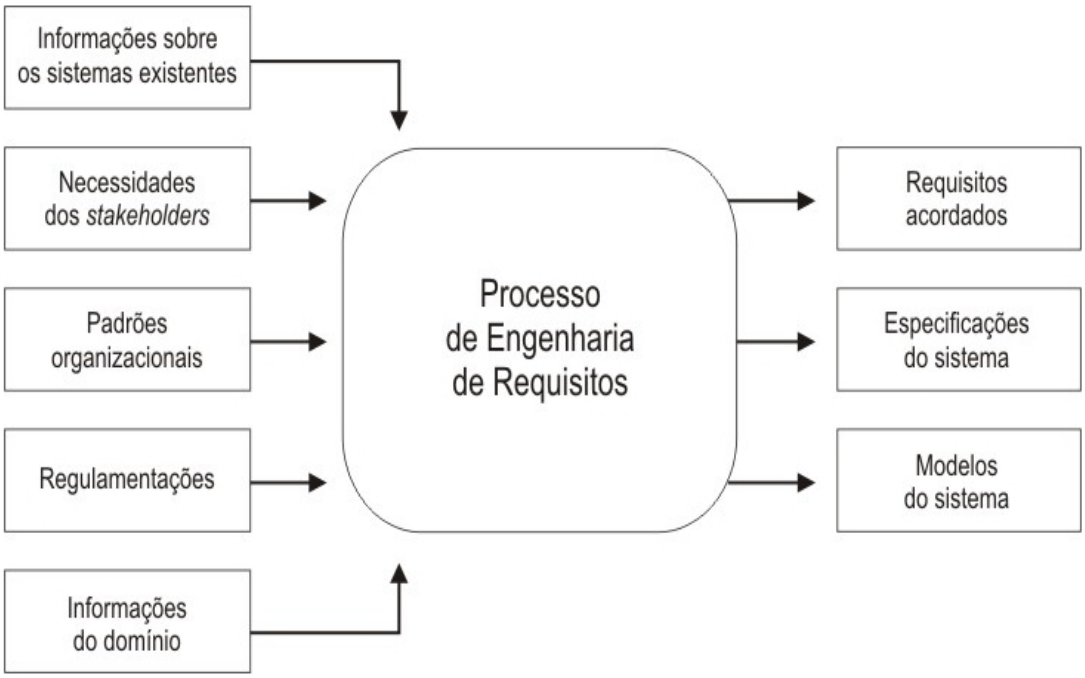

Figura 5 - Processos de Engenharia de Requisitos [https://www.vivaolinux.com.br/artigo/SIGERAR-Sistema-de-Gerenciamento-de-Requisitos acessado em 13/12/2016]

[PRES2011] destaca que algumas funções da Engenharia de Requisitos ocorrem em paralelo e todas são adaptadas às necessidades do projeto, tentam definir o que o cliente deseja e estabelecem uma fundação sólida para o projeto e construção do software.

2.2.5. DOCUMENTOS DE REQUISITOS

O resultado principal do processo de Engenharia de Requisitos é um documento de Especificação de Requisitos de Software (SRS). Apesar do documento de especificação de requisitos ser de fundamental importância a um produto de software de qualidade, muitos usuários, desenvolvedores e gerentes não entendem a necessidade e acreditam que é importante começar o mais cedo possível a codificação.

Um dos motivos para isso é o fato de que uma especificação de requisitos feita com qualidade custa tempo e consequentemente dinheiro, além de ser necessária competência para a criação. De acordo com pesquisas e literaturas referentes a requisitos, a ausência de uma especificação de requisitos de qualidade, custa ainda mais tempo e dinheiro nas fases posteriores (codificação), sendo de suma importância que os Engenheiros de Software insistam sempre na elaboração de uma especificação de requisitos.

Uma especificação de requisitos é um documento criado quando uma descrição detalhada de todos os aspectos do software a ser construído deve ser especificada antes de o projeto começar [PRES2011].

Existem diversas formas descritas na literatura que ocasionam controvérsia e ambiguidade para gerar tal documentação. Uma estrutura proposta por [PRES2011] é apresentada abaixo:

- Sumário
- Histórico de revisão
- Introdução
    - Propósito
    - Convenções do documento

    - Público-alvo e sugestões de leitura
    - Escopo do projeto
    - Referências
- Descrição geral
    - Perspectiva do produto
    - Características do produto
    - Classes de usuários e características
    - Ambiente operacional
    - Restrições de projeto e implementação
    - Documentação para usuários
    - Hipóteses e dependências
- Características do sistema
- Requisitos de interfaces externas
    - do Usuário
    - de Hardware
    - de Software
    - de Comunicação
- Outros requisitos não funcionais
- Outros requisitos
- Glossário
- Modelos de análise
- Lista de problemas

Uma outra estrutura proposta pelo, e amplamente utilizada é fornecida pelo IEEE (IEEE/ANSI 830-1993).

É importante salientar que outros artefatos, além dos requisitos funcionais e não funcionais, poderão ser associados ao documento de especificação, como:

- Casos de uso: documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do

mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz.

- Regras de negócio: definem como o seu negócio funciona, podem abranger diversos assuntos como suas políticas, interesses, objetivos, compromissos éticos e sociais, obrigações contratuais, decisões estratégicas, leis e regulamentações entre outros.

Matriz de rastreabilidade: tem por objetivo permitir aos envolvidos no projeto descobrir e entender o relacionamento entre cada requisito e suas fontes, além da associação com produtos do projeto. Permite identificar a origem de funcionalidades, associá-las à execução de testes e verificar/auxiliar no impacto de mudanças no projeto.

2.2.6. QUALIDADE DE REQUISITOS

Quando os requisitos são especificados no processo de desenvolvimento de software é de fundamental importância a garantia da qualidade. A norma internacional IEEE 830-1998 elenca boas práticas para garantir que os requisitos especificados obtenham maior nível de qualidade.

A seguir são apresentadas 8 propriedades descritas pela norma citada que visa alcançar um nível de qualidade aceitável.

- Correta
- Sem ambiguidades
- Completa
- Consistente
- Priorização por importância e/ou estabilidade
- Verificável (testes)
- Modificável (sem alterar atributos)
- Rastreável (Documentação/Controle de Versões)

- **Correta**

Uma especificação de requisitos é considerada correta se cada requisito expresso nela são requisitos que o software deva atender. Não há nenhuma ferramenta ou procedimento que garante este item.

A especificação de requisitos deve ser comparada com qualquer especificação aplicável superior, tal como uma especificação de requisitos de sistema, com a documentação de outro projeto, e com outras normas aplicáveis, garantindo que todos estes itens estejam em concordância.

Alternativamente, o cliente pode determinar se a especificação reflete corretamente suas necessidades reais. A rastreabilidade torna este procedimento mais fácil e menos propenso a erros.

- **Sem Ambiguidades**

Uma especificação de requisitos é isenta de ambiguidades se cada requisito expresso nela tem apenas uma interpretação. No mínimo, isto requer que cada característica do produto final seja descrita usando um termo único.

Nos casos em que um termo utilizado, num contexto particular, tenha múltiplos significados, este termo deve ser incluído em um glossário onde o seu significado é determinando de maneira mais específica.

A especificação de requisitos é uma parte importante do ciclo de vida do software e é usada na implementação, monitoramento, verificação e validação.

Assim, a especificação deve estar livre de ambiguidades, tanto para aqueles que criam quanto para aqueles que a usam.

No entanto, estes grupos muitas vezes não têm o mesmo conhecimento e, portanto, não descrevem os requisitos de software da mesma maneira.

Algumas representações que melhoram a especificação de requisitos para o desenvolvedor podem ser ruins, na medida em que diminuem a compreensão para o usuário e vice-versa.

Os itens abaixo apresentam recomendações de como evitar ambiguidades nos requisitos de software.

Armadilhas da linguagem natural: Requisitos são geralmente escritos em linguagem natural (por exemplo, em Português ou Inglês). A linguagem natural é inerentemente ambígua. Uma especificação de requisitos escrita em linguagem natural deve ser revista e validada para que seja identificado o uso ambíguo da linguagem e assim possa ser corrigido.

Linguagens de especificação de requisitos: Uma maneira de evitar a ambiguidade inerente à linguagem natural é escrever a especificação em um determinado idioma de especificação de requisitos. Assim, os processadores de idiomas automaticamente detectam muitos erros lexicais, sintáticos e semânticos.

Uma desvantagem na utilização de tais idiomas é o tempo necessário para aprendê-los. Além disso, muitos usuários não técnicos terão dificuldades em entender a linguagem.

Outro ponto importante é que essas linguagens tendem a ser melhores em expressar certos tipos de necessidades e resolver certos tipos de sistemas. Assim, eles podem influenciar os requisitos de maneiras sutis.

Ferramentas de representação: Em geral, os métodos de requisitos, linguagens e as ferramentas que lhes dão suporte são classificados em três categorias gerais.

Abordagens orientadas a objetos organizam os requisitos em termos de objetos do mundo real, seus atributos e os serviços realizados por esses objetos.

Abordagens baseadas em processos organizam os requisitos em hierarquias de funções que se comunicam através de fluxos de dados.

Abordagens comportamentais descrevem o comportamento externo do sistema em termos de alguma noção abstrata (como cálculo de predicados), funções matemáticas, ou máquinas de estado.

O grau em que tais ferramentas e métodos podem ser úteis na preparação de uma especificação de requisitos depende do tamanho e complexidade do sistema esperado.

A melhor forma de usar qualquer uma dessas abordagens é manter as descrições em linguagem natural. Dessa forma, os clientes que não estão familiarizados com as notações podem entender a especificação.

- **Completa**

Uma especificação de requisitos é completa se inclui os seguintes elementos:
Todos os requisitos devem ser significativos, seja em matéria de funcionalidade, desempenho, restrições de design, atributos ou interfaces externas. Em particular, quaisquer exigências externas impostas por um sistema ou especificação devem ser reconhecidas e tratadas.

Existe uma definição das respostas do software para todas as classes realizáveis de dados de entrada em todas as situações. É de extrema importância especificar as respostas para os valores de entrada válidos e inválidos.

Devem estar definidos os rótulos completos e referências a todas as figuras, tabelas e diagramas na especificação.

Utilização de ASDs: Qualquer especificação que use a frase "a ser determinado" (ASD) não é uma especificação de requisitos completa.
O ASD é, no entanto, por vezes, necessário e deve ser acompanhado por:

Uma descrição das condições que causam o ASD (por exemplo, uma resposta não é conhecida), de modo que a situação possa ser resolvida;
A descrição do que deve ser feito para eliminar o ASD, o que é responsável pela sua eliminação, e quando ele deve ser eliminado.

- **Consistente**

Consistência refere-se à consistência interna. Se uma especificação de requisitos de software não está de acordo com algum documento de nível superior, como uma especificação de requisitos de sistema, então ela não é considerada consistente.

Uma especificação é internamente consistente se nenhum subconjunto de requisitos individuais nela descritos estão em conflito. Os três tipos de conflitos prováveis em um Software Requirement Specification (SRS) são:

As características específicas de objetos do mundo real podem entrar em conflito. Por exemplo, um requisito pode afirmar que todas as luzes devem ser verdes enquanto outro pode afirmar que todas as luzes devem ser azuis.
Pode haver um conflito lógico ou temporal entre duas ações. Por exemplo, um requisito pode especificar que o programa irá adicionar duas entradas e outro pode especificar que irá multiplicá-los.

Dois ou mais requisitos podem descrever o mesmo objeto do mundo real, mas usam termos diferentes para aquele objeto. Por exemplo, uma entrada do usuário pode ser chamada de “imediata” em um requisito e “dependente” em outro. O uso de terminologia padrão e definições promovem a consistência.

- **Classificada por importância e/ou estabilidade**

Uma especificação de requisitos de software é classificada por importância e/ou estabilidade se cada exigência tem um identificador para importância ou estabilidade de um requisito em particular.

Tipicamente, todos os requisitos que se relacionam com um produto de software não são igualmente importantes. Alguns requisitos podem ser essenciais, especialmente para aplicações críticas, enquanto outros podem ser desejáveis.

Cada requisito deve ser identificado para que essas diferenças sejam claras e explícitas.

Identificar os requisitos dessa forma ajuda o cliente, fazendo com que ele seja mais cuidadoso considerando para cada necessidade, e ajuda também os desenvolvedores a tomar decisões de projeto corretas e dedicar níveis adequados de esforço para as diferentes partes do produto de software.

Grau de estabilidade: Um método de identificação de requisitos usa a dimensão de estabilidade. A estabilidade pode ser expressa em termos do número de mudanças esperadas para qualquer exigência, com base na experiência ou conhecimento dos eventos, que afetam a organização, as funções, e as pessoas suportadas pelo sistema de software.

Grau de necessidade: Outra forma de classificar os requisitos é distinguir classes de requisitos como essencial, condicional e opcional.
Essencial: Implica que o software não será aceitável, a menos que estes requisitos sejam fornecidos da maneira acordada.

Condicional: implica que estes são requisitos que melhoram o produto de software, mas não o tornam inaceitável se estiverem ausentes.
Opcional: implica uma classe de funções que podem ou não podem ser benéficos.

- **Verificável**

Uma especificação de requisitos de software é verificável se cada requisito expresso nela é verificável. Determinando que, um requisito é verificável se existe algum processo finito com o qual uma pessoa ou máquina pode verificar que o produto de software atende a exigência.

Em geral, qualquer requisito ambíguo não é verificável. Requisitos não verificáveis incluem declarações como "funciona bem", "boa interface humana", e "normalmente acontece". Estes requisitos não podem ser verificados porque é impossível definir os termos "bem", "boa" ou "normalmente".

Uma afirmação do tipo "o programa não deverá entrar em loop infinito" é não verificável também porque o teste desta qualidade é teoricamente impossível. Um exemplo de um requisito verificável poderia ser: “A Saída do programa deve ser produzido dentro de 20 s depois do evento em 60% do tempo, e deve ser apresentada dentro de 30 s depois do evento em 100% do tempo”.

Esta afirmação pode ser verificada porque ele usa termos concretos e quantidades mensuráveis. Se um método não pode ser concebido para determinar se o requisito atende a uma necessidade específica, então este requisito deve ser removido ou revisto.

- **Modificável**

Uma especificação de requisitos de software é modificável se, e somente se, sua estrutura e estilo são tais que, quaisquer mudanças nos requisitos podem ser feitas facilmente, completa e consistente, mantendo a estrutura e estilo. Isso geralmente requer:

Ter uma organização coerente e fácil de usar com uma tabela de conteúdos, um índice e referências explícitas.

Não ser redundante (ou seja, o mesmo requisito não deve aparecer em mais de um lugar na especificação).

Cada requisito é expresso separadamente, ao invés de misturados com outros requisitos.

Importante destacar que a redundância em si não é um erro, mas pode facilmente levar a erros.

Redundância pode ocasionalmente ajudar a fazer uma especificação mais legível, mas um problema pode surgir quando o documento redundante é atualizado. Por exemplo, um requisito pode ser alterado em somente um dos lugares onde ele aparece.

A especificação, em seguida, torna-se inconsistente. Sempre que a redundância é necessária, a especificação deve incluir explícitas referências cruzadas para torná-la modificável.

- **Rastreável**

Uma especificação de requisitos é rastreável se a origem de cada um de seus requisitos é clara e facilita a referência de cada requisito no desenvolvimento ou documentação futura.

Os dois tipos de rastreabilidade recomendadas são:

Rastreabilidade para trás (Backward): às fases anteriores de desenvolvimento. Isso depende de cada requisito e sua referência explícita a fonte em documentos anteriores.

Rastreabilidade para frente (Forward): a todos os documentos gerados pela especificação. Isso depende de cada requisito ter um único nome ou número de referência. A rastreabilidade para frente é especialmente importante quando o produto de software entra em operação e na fase de manutenção.

Como documentos de código e de concepção são modificados, é essencial determinar o conjunto completo de requisitos que podem ser afetados por essas modificações.

3. MÉTODO COSMOD-RE

3.1. DEFINIÇÃO DO MÉTODO

Segundo o autor Pohl Klaus, o COSMOD-RE apoia o desenvolvimento de requisitos e artefatos de arquitetura para sistemas de software em várias camadas de abstração. Os blocos de construção fundamentais do método COSMOD-RE são:

- As quatro camadas de abstração.
- Os quatro tipos de artefatos de desenvolvimento definidos em cada camada de abstração (objetivos, cenários, requisito orientado a solução e arquitetura).
- Os três processos de design para desenvolver os requisitos e artefatos de arquitetura. Cada sub-processo consiste em cinco sub-processos.

O principal objetivo do COSMOD-RE (Método de desenvolvimento baseado em Objetivos e Cenários - s**C**enario and g**O**al based **S**ystem development **M**eth**OD** - **RE**quirements) é apoiar o desenvolvimento interligado de requisitos e artefatos de arquitetura para sistemas de software.

Como mencionado acima, o COSMOD-RE consiste em três blocos de construção:

Hierarquia de quatro camadas de abstração: as camadas de sistema, de decomposição funcional, de particionamento de hardware e software, e a de implantação.

Quatro tipos de artefatos básicos em cada camada: Em cada camada de abstração, o método define quatro tipos de artefatos: objetivos, cenários, requisito orientado a solução e arquitetura.

Três processos de design e cinco sub-processos: oferece três processos de design: os designs de níveis de sistema, de função e de hardware e software.

Cada um dos três processos de design concentra-se em duas camadas de abstração próximas e consiste em cinco sub-processos.

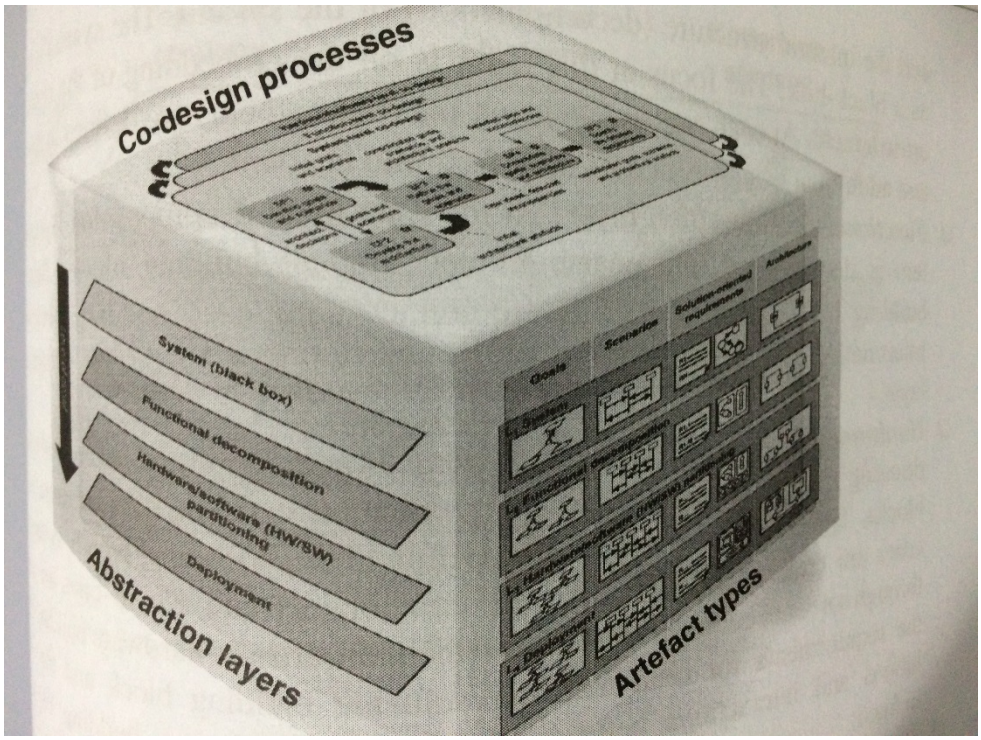

Figura 6 - Blocos de construção do método COSMOD-RE [POHLKLAUS]

3.2. AS QUATRO CAMADAS DE ABSTRAÇÃO

Cada camada de abstração tem um foco bem definido e define um ponto de vista específico no sistema. Requisitos e artefatos de arquitetura são definidos.

Para ilustrar as características essenciais e alcance dos artefatos de requisitos definidos em cada camada de abstração são utilizados modelos de requisitos comportamentais (um subconjunto dos artefatos de requisitos normalmente são definidos em cada camada). Para os artefatos de arquitetura são utilizados modelos de arquitetura estruturais simplificados.

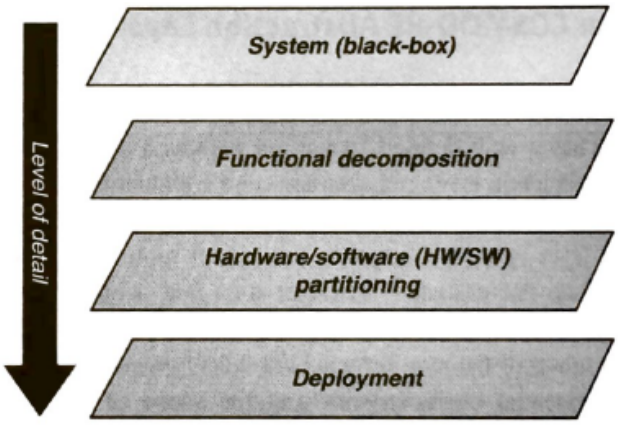

Figura 7 - Camadas de Abstração [POHLKLAUS]

Camada Sistema: A camada sistema (a mais alta camada de abstração) define os requisitos e artefatos de arquitetura independentemente da tecnologia de implementação e estrutura interna (decomposição) do sistema.

O sistema é considerado como uma caixa preta. O foco nesta camada é sobre a incorporação do sistema no seu ambiente. Em cada camada sucessiva, os requisitos são definidos em mais detalhes e preocupações adicionais de design são tomadas em consideração.

As interações entre o sistema e seu ambiente são definidas em termos de cenários. Os objetivos ou propriedades do sistema de alto nível são definidos em termos de objetivos. Requisitos detalhados, tais como o comportamento externamente visível do sistema são especificados usando modelos de requisitos orientado a solução (por exemplo, diagramas UML, máquina de estado, casos de uso).

O sistema interage com o seu ambiente por meio de interfaces definidas. As interfaces do sistema definidas na camada do sistema especificam a incorporação do sistema em seu ambiente.

Estas interfaces são definidas independentemente de uma solução técnica. Para apoiar a compreensão do sistema e suas interações com o meio ambiente, sistemas externos e atores devem ser incluídos no modelo de arquitetura.

No modelo simplificado representado na imagem abaixo, um sistema de freio

com duas interfaces de entrada e uma interface de saída foi definido. Um sistema de freio típico compreende a mecânica, hidráulica, eletrônicos e componentes de software. Do ponto de vista da camada de sistema (caixa preta) todos os detalhes estão escondidos e somente as interfaces abstratas do sistema de freio são definidas.

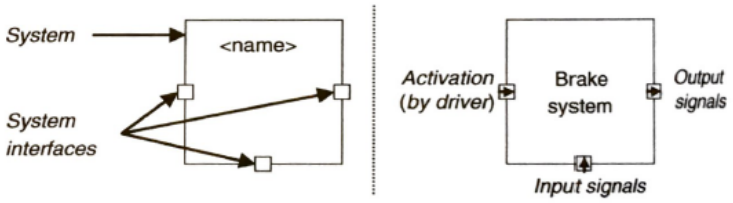

Figura 8 – Interfaces [POHLKLAUS]

Camada de Decomposição Funcional: Nesta camada o sistema é decomposto em blocos de construção lógicos de granularidade de alto nível. Além dos próprios blocos de construção, os requisitos para cada bloco de construção, os relacionamentos e as interações entre os blocos de construção são definidos.

A preocupação principal da segunda camada é a decomposição funcional do sistema. Cada componente lógico que é definido nesta camada representa uma unidade de funcionalidade coerente e por isso é chamado de componente funcional.

Os requisitos e artefatos de arquitetura definidos nessa camada são independentes de uma tecnologia de aplicação particular, como em Análise de Sistemas Essenciais[3].

Por exemplo, a funcionalidade do sistema é definida independentemente de um particionamento de hardware e software. Os artefatos definidos nesta camada permanecem assim, mesmo se a tecnologia de implementação (relativamente estável) nas camadas inferiores sofrerem mudanças.

[3] A Análise Essencial propõe o particionamento do sistema por eventos. A rigor, o valor de um sistema está na sua capacidade de responder com eficácia a todos os estímulos a que for submetido. Assim, um sistema é construído para responder a estímulos. A cada estímulo, o sistema deve reagir produzindo uma resposta predeterminada. A expressão Essential Analysis, traduzida por Análise Essencial, foi proposta em 1984 por McMenamim e Palmer para refletir a introdução dos novos conceitos que estavam sendo incorporados à Análise Estruturada clássica. Fonte https://pt.wikipedia.org/wiki/An%C3%A1lise_essencial acessado em 30 de maio de 2016.

Além disso, a definição independente de tecnologia suporta a reutilização dos requisitos e artefatos de arquitetura através das fronteiras do sistema.

Na camada de decomposição funcional, requisitos (objetivos, cenários e requisitos orientado a solução) são definidos para cada componente funcional. Ao definir os requisitos para os componentes funcionais do sistema, os requisitos definidos no nível acima são considerados.

Os artefatos de arquitetura neste nível definem os componentes lógicos das interfaces lógicas do sistema e conexões lógicas entre os componentes. A arquitetura abstrai, assim, a partir de qualquer diferenciação entre os componentes de hardware e software, bem como de dispositivos físicos, interfaces físicas, e redes de comunicação físicas.

Assim, a arquitetura lógica (funcional) define uma estrutura para o problema e é, além dos requisitos, uma contribuição essencial para a definição de uma arquitetura de sistema mais concreta na camada de particionamento de hardware e software.

O exemplo abaixo mostra três componentes lógicos de um sistema de freio: pedal do travão, controle do freio e os freios das rodas.

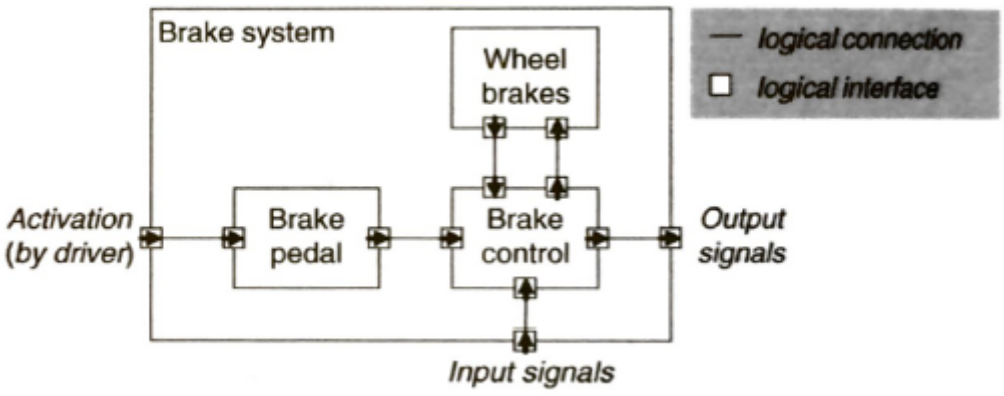

Figura 9 - Componentes Funcionais [POHLKLAUS]

Além da arquitetura lógica (funcional), requisitos de qualidade devem ser definidos para a arquitetura nesta camada. Exemplos de requisitos de qualidade

são a adaptabilidade da funcionalidade do sistema ou a reutilização dos componentes funcionais.

Camada de Particionamento Hardware e Software: Nesta camada o sistema é decomposto em blocos de construção de hardware e software. Assim, as tomadas de decisões dessa camada são sobre quais propriedades do sistema são realizadas por meio de componentes de hardware e quais são realizadas através de componentes de software.

São definidos os requisitos para cada bloco de construção de hardware e software e os relacionamentos e interações entre os blocos de hardware e software de construção.

Os requisitos na camada de particionamento de hardware e software são definidos com base nos requisitos especificados na camada de decomposição funcional e estão associados aos componentes de hardware e software definidos nesta camada. Além disso, os requisitos adicionais são extraídos e definidos da decomposição escolhida do sistema em componentes de hardware e de software.

A decomposição do sistema considerada na camada de particionamento de hardware e software define uma arquitetura de granularidade de alto nível que consiste em componentes de hardware e software, suas interfaces e conectores que definem os canais de comunicação entre os componentes de hardware e software. Ao definir esta arquitetura, tipicamente requisitos de qualidade, tais como desempenho e custo são considerados.

Componentes de hardware e software que pertencem a plataforma do hardware e software subjacentes do sistema, tais como o sistema operacional, drivers de dispositivo padrão, unidades de processamento, entre outros, não são definidos na camada de particionamento de hardware e software.

Camada de Implantação: A implantação do sistema na plataforma de hardware e software (rede de unidades físicas) é definida. A definição da implantação inclui

a decisão sobre qual componente de software e em que unidade física será executado.

Os requisitos e artefatos de arquitetura definidos na camada de particionamento de hardware e software estão agrupados para formar unidades físicas, tais como unidades de controle eletrônico de um veículo ou de uma aeronave por exemplo.

A imagem abaixo ilustra tal agrupamento. O sistema representado na figura é um sistema distribuído que consiste em duas unidades de controle eletrônico (ECU1 e ECU2). Cada ECU está equipada com o sistema operacional na (RTOS1 e RTOS2). Os componentes SW1 e Sensor1 foram atribuídos (implantado) para ECU1 e os componentes SW2, SW3 e atuador1 foram designados para ECU2.

A conexão entre SW1 e SW2 é realizada por um barramento de comunicação que liga ECU1 e ECU2.

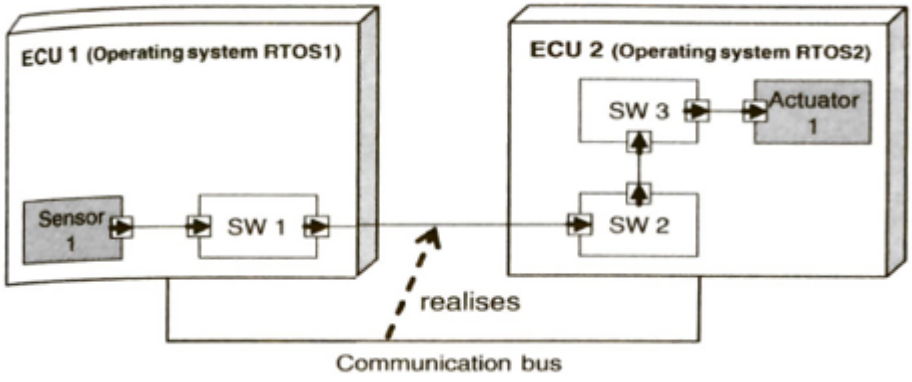

Figura 10 - Exemplo de Implantação de HW e SW [POHLKLAUS]

Dependendo do tipo de projeto, um dos três casos seguintes podem ser aplicados:

Os componentes de hardware e software são implantados na topologia existentes de unidades físicas que é predefinida e permanece inalterada.

Uma topologia existente de unidades físicas é modificada durante a implantação. Por exemplo, pode ser adicionada uma nova unidade de controle.

Uma nova topologia de unidades física é criada e, assim, não é usada topologia predefinida.

A implantação dos componentes de software e hardware para a topologia de unidades físicas pode exigir refinamento e ajuste dos requisitos definidos na camada de particionamento de hardware e software. Por exemplo, pode ser necessário alterar uma exigência devido às propriedades específicas de um componente de hardware concreto pertencente à plataforma hardware e software (tais como o tempo disponível de cálculo, a memória, largura de banda de rede).

Desenvolvimento de cima para baixo (top-down) não são aplicados: Embora o método COSMOD-RE promova uma hierarquia de quatro camadas de abstração, o método não impõe um processo de desenvolvimento de cima para baixo. Um rigoroso processo de desenvolvimento top-down exigiria que os requisitos e artefatos de arquitetura fossem definidos primeiramente e exclusivamente na camada do sistema e, em seguida, detalhado na camada de decomposição functional.

Em contraste, o método apoia o desenvolvimento de requisitos e artefatos de arquitetura através das camadas de abstração, ou seja, apoia o *top-down*, *bottom-up*, bem como o desenvolvimento *middle-out* e alinhamento dos requisitos e artefatos de arquitetura.

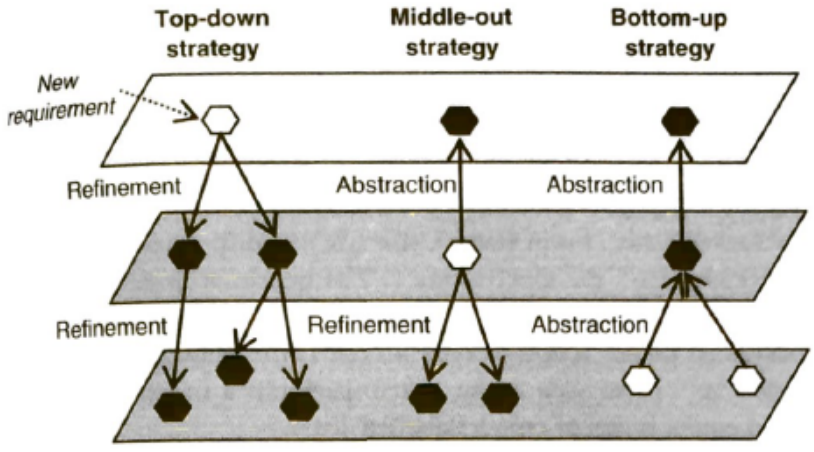

Figura 11 - Estratégias de decomposição [POHLKLAUS]

3.3. OS QUATRO TIPOS DE ARTEFATOS

A imagem abaixo fornece uma visão geral dos diferentes tipos de artefatos definidos em cada camada de abstração. Como representado, o COSMOD-RE diferencia, em cada camada de abstração, quatro tipos básicos de artefatos:

- Objetivos
- Cenários
- Requisito orientado a solução
- Arquitetura

Nota-se que os tipos de artefatos utilizados em um projeto específico deve ser selecionado de acordo as necessidades específicas do projeto. Em outras palavras, nem todos os artefatos devem ser usados dentro de um projeto ou em cada camada de abstração.

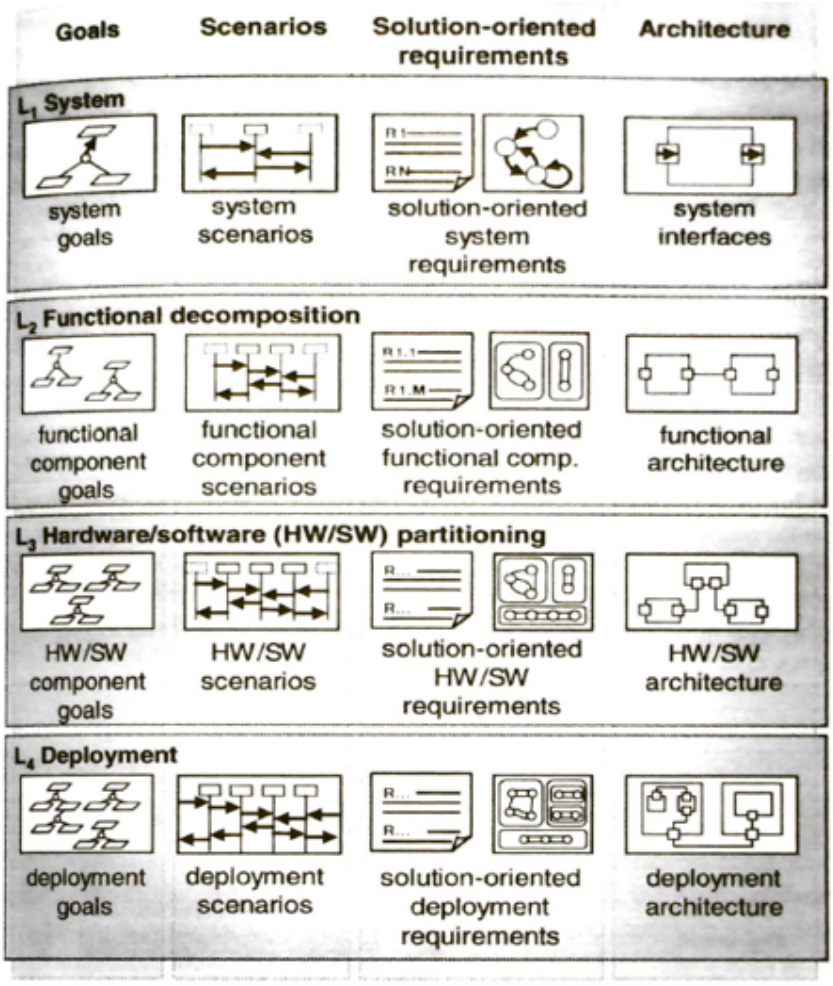

Figura 12 - Visão geral das camadas de abstração e tipos de artefatos [POHLKLAUS]

Objetivos: Os objetivos são definidos em cada uma das quatro camadas de abstração.

Por exemplo, o objetivo geral do sistema é definido na camada do sistema, enquanto que um objetivo específico para um componente de hardware é definido na camada de particionamento de hardware e software. Durante a transição de uma camada de abstração mais alta para uma camada de abstração mais baixa, os objetivos para os componentes na camada de abstração mais baixa são definidos considerando os objetivos definidos pela camada de abstração mais alta.

Além disso, novos objetivos que não resultaram de refinamento dos objetivos de alto nível pode ser identificado na camada de abstração inferior, conforme a imagem abaixo:

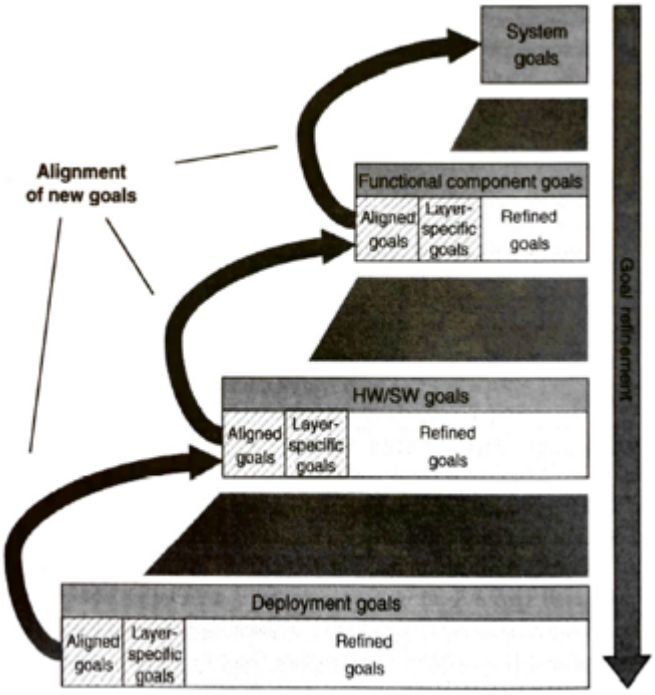

Figura 13 - Definições dos Objetivos nas Camadas de abstração [POHLKLAUS]

Se um novo objetivo é identificado, um dos quatro casos básicos podem ser aplicados:

Caso 1: O novo objetivo pode estar relacionado a um super-objetivo definido na camada de abstração mais elevada.

Caso 2: O novo objetivo indica uma abertura na camada de abstração mais elevada. Depois de definir o super-objetivo até agora negligenciado, o novo objetivo está relacionado a este super-objetivo.

Caso 3: O novo objetivo não pode ser relacionado com um super-objetivo na camada de abstração mais elevada, porque é irrelevante para o sistema.

Caso 4: O novo objetivo não pode ser relacionado com um super-objetivo na camada de abstração mais elevada, porque ele é específico para a camada de abstração mais baixa.

Por exemplo, os objetivos específicos de tecnologia que dependem da tecnologia escolhida só deverão ser definidos na camada de abstração onde a escolha da tecnologia é manifestada. Do mesmo modo, os objetivos que dependem de um sistema particular de decomposição devem ser definidos somente na camada onde a decomposição é definida.

Restrições relativas a solução desejada (por exemplo, imposto pelo cliente, uma lei, a infra-estrutura técnica) deve ser definida como restrições de design explícitas e não escondida nas definições de objetivo. Portanto, um objetivo definido em uma camada de abstração específica não deve prescrever uma solução específica, como uma arquitetura específica ou uma decomposição específica nas camadas de abstração abaixo.

As partes interessadas obtem e documentam novos objetivos de componentes funcionais que não têm qualquer relação com um objetivo do sistema. Se um novo objetivo de componente funcional é identificado, os quatro casos para delinear são aplicáveis.

Cenários: O COSMOD-RE emprega interação de cenários da camada de abstração de sistema com os cenários internos do sistema das camadas de abstração inferiores. Cenários de interação permitem a descrição e análise das interações que ocorrem entre o sistema e seus atores externos, enquanto que os cenários internos consideram e definem as interações entre os componentes do sistema.

Cenários de sistema interno pode ser utilizado para as três camadas inferiores, para apoiar a identificação de relações entre os componentes ou ainda a identificar os próprios componentes.

Cenários de sistema que definem as interações do sistema com o seu ambiente são sucessivamente refinados nas camadas de abstração mais baixas. Quando um cenário é refinado, a instância representando o sistema é decomposta em um conjunto de componentes, por exemplo componentes funcionais na camada de decomposição funcional ou componentes de hardware e software na camada de particionamento de hardware e software.

No cenário refinado, além das interações já definidas nos cenários nas camadas de abstração mais elevadas, são adicionadas interações entre as instâncias que resultam da decomposição. Essas novas interações são obrigadas a cumprir o cenário definido na camada de abstração mais elevada.

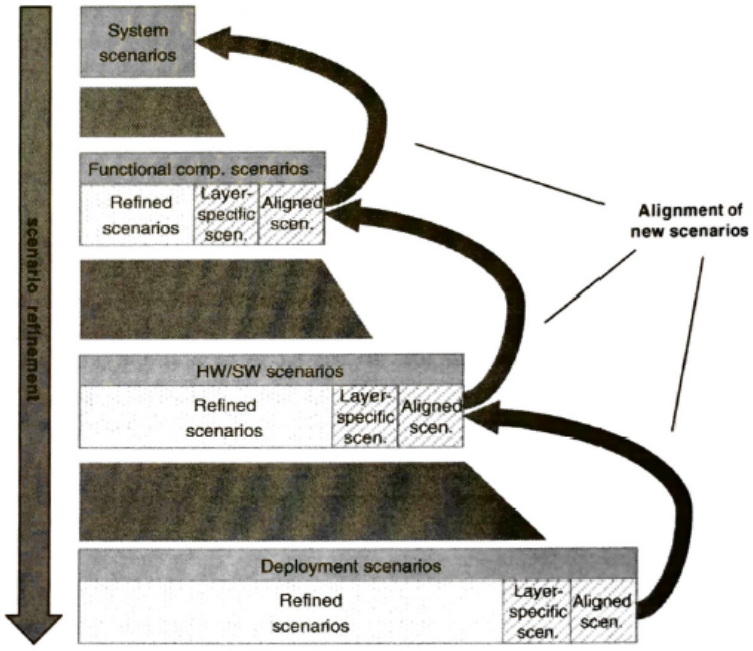

Figura 14 - Refinamento e Consolidação de Cenários nas camadas de abstração [POHLKLAUS]

O cenário na camada de abstração mais elevada pode ser refinado em um ou mais cenários em uma camada de abstração inferior. Por exemplo, pode haver diferentes formas de satisfazer um cenário de sistema por uma sequência de interações internas do sistema. Nesse caso, na camada de abstração menor, um

cenário principal e diversos cenários alternativos são definidos ao refinar o cenário do sistema.

Além de refinar os cenários definidos para as camadas de abstração mais elevadas, interações adicionais (por exemplo, com o meio ambiente), bem como novos cenários podem ser identificados nas camadas de abstração mais baixa.

Se um cenário novo é identificado em uma camada de abstração inferior, um dos seguintes casos podem ser aplicados:

O novo cenário pode estar relacionado a um cenário que já está definido na camada de abstração mais elevada.

O novo cenário indica um cenário esquecido na camada de abstração mais elevada. Depois de definir o cenário esquecido, os dois cenários são ligados um ao outro por uma relação refinada.

O novo cenário não pode ser relacionado com um cenário de uma camada de abstração mais elevada, porque o cenário é irrelevante para o sistema.

O novo cenário não pode ser relacionado com um cenário na camada de abstração mais elevada, porque ele é específico para a camada de abstração mais baixa.

Requisito Orientado a Solução: Requisitos são detalhados e definidos em cada camada de abstração. Requisitos detalhados compreendem requisitos funcionais, tais como funções, dados e comportamento, bem como os requisitos de qualidade, tais como desempenho, segurança ou requisitos de segurança e restrições. Os requisitos orientados a solução podem ser especificados usando linguagem natural, bem como a utilização de modelos como fluxo de dados, de classe, ou de estado.

A granularidade e escopo de um requisito orientado a solução dependem da camada de abstração em que os artefatos de requisitos são definidos. Requisitos funcionais e de qualidade que afetam todo o sistema são definidos na camada do

sistema. Requisitos relativos aos componentes funcionais individuais são definidos na camada de decomposição funcional. Requisitos para os componentes de hardware e software são definidos na camada de particionamento de hardware e software. Na camada de implantação, os requisitos para as unidades físicas são definidos com base nos requisitos de componentes de hardware e de software definidos na camada acima.

Os requisitos de comportamento em uma camada de abstração específica definem o comportamento externamente visível do sistema (camada do sistema), um componente funcional (camada de decomposição funcional), um componente de hardware ou software (camada de particionamento de hardware e software), ou uma unidade física da rede (camada de implantação).

Nota-se que a realização do comportamento exigido pode ser especificado usando artefatos de arquitetura. Em contraste com os requisitos comportamentais, os artefatos de arquitetura não estão restritos ao comportamento externo na respectiva camada, isto é, eles podem adicionalmente definir o comportamento interno do sistema, o componente funcional, componente de hardware e software, ou uma unidade física de rede.

Arquitetura: O método COSMOD-RE emprega modelos de arquitetura estruturais a fim de documentar a decomposição do sistema nas quatro camadas de abstração. Além disso, o comportamento interno dos componentes de arquitetura em cada camada pode ser definido em detalhe usando máquinas de estados.

O modelo de arquitetura estrutural consiste em componentes, interfaces e conectores. Os componentes possuem interfaces definidas e são ligados um ao outro por meio de conectores. Cada uma das quatro camadas de abstração tem um certo foco e escopo e, portanto, considera um tipo específico de decomposição do sistema.

3.4. OS TRÊS PROCESSOS DE DESIGN

O método oferece três processos de design distintos que estruturam o processo de desenvolvimento ao longo das quatro camadas de abstração.

Cada um dos três processos de design focam principalmente em duas camadas de abstração. Por exemplo, o processo de design do sistema concentra-se sobre o desenvolvimento dos artefatos na camada do sistema e artefatos na camada de decomposição funcional.

Além disso, cada processos de design sobrepõe parcialmente outro processo de design em relação a uma camada de abstração específica. Os objetivos dos três processos de design pode ser caracterizado resumidamente da seguinte forma:

Nível de design do sistema: O principal objetivo deste processo de design é desenvolver os requisitos do sistema, juntamente com uma decomposição funcional inicial do sistema.

Nível de design de função: O âmbito deste processo de design é a definição dos requisitos de componentes funcionais e um particionamento de hardware e software inicial.

Nível de design de hardware e software: Este processo de design tem como objetivo principal definir os requisitos de hardware e software, juntamente com uma implantação inicial dos componentes de hardware e software para a plataforma de hardware e software.

Cada processo de design apoia o desenvolvimento interligado de requisitos e artefatos de arquitetura. Por exemplo, os requisitos do sistema e de arquitetura funcional, o processo de design do sistema suporta o desenvolvimento interligado dos requisitos do sistema e a arquitetura funcional. A relação entre os tipos de artefatos e os três processos de design é mostrada na imagem abaixo.

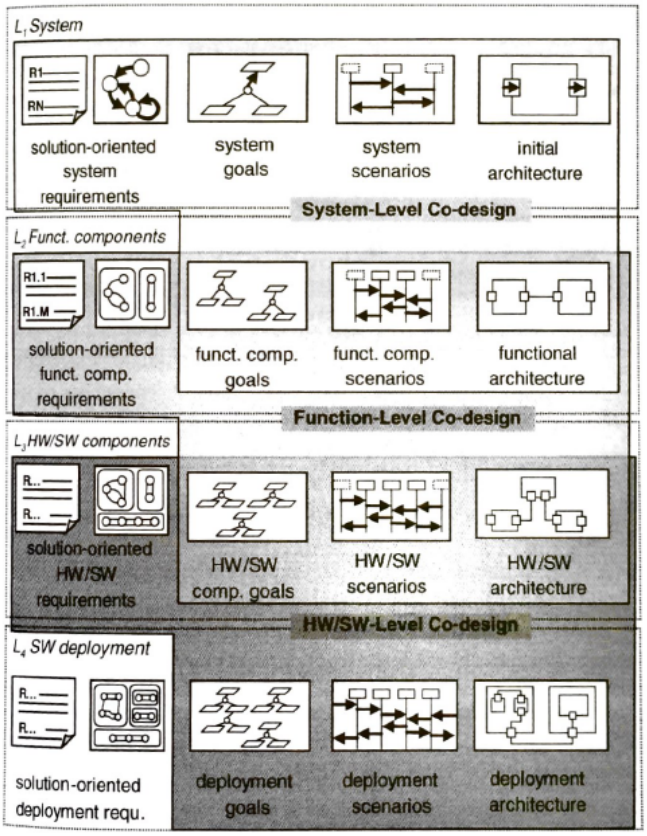

Figura 15 - Artefatos e os três processos de design [POHLKLAUS]

Se os processos de design são realizados por equipes diferentes, e os resultados parciais do processo de design no nível do sistema estão disponíveis, o processo de design de nível de função pode ser iniciado com base nestes resultados parciais. Uma vez que, neste momento, o processo de criação de nível de sistema ainda não foi terminado, os dois processos de concepção são executadas em paralelo.

Além disso, várias instâncias do processo de design em nível de função pode ser realizada em paralelo. Após definir a arquitetura funcional de alto nível no processo de design em nível de sistema, várias instâncias do processo de design em nível de função pode ser iniciada, cada instância focando um componente funcional diferente.

3.5. OS CINCOS SUB-PROCESSOS DE DESIGN

O método COSMOD-RE fornece cinco sub-processos inter-relacionados para cada processo de design. A imagem abaixo mostra uma visão geral dos cinco sub-processos e os fluxos de informação principal entre eles.

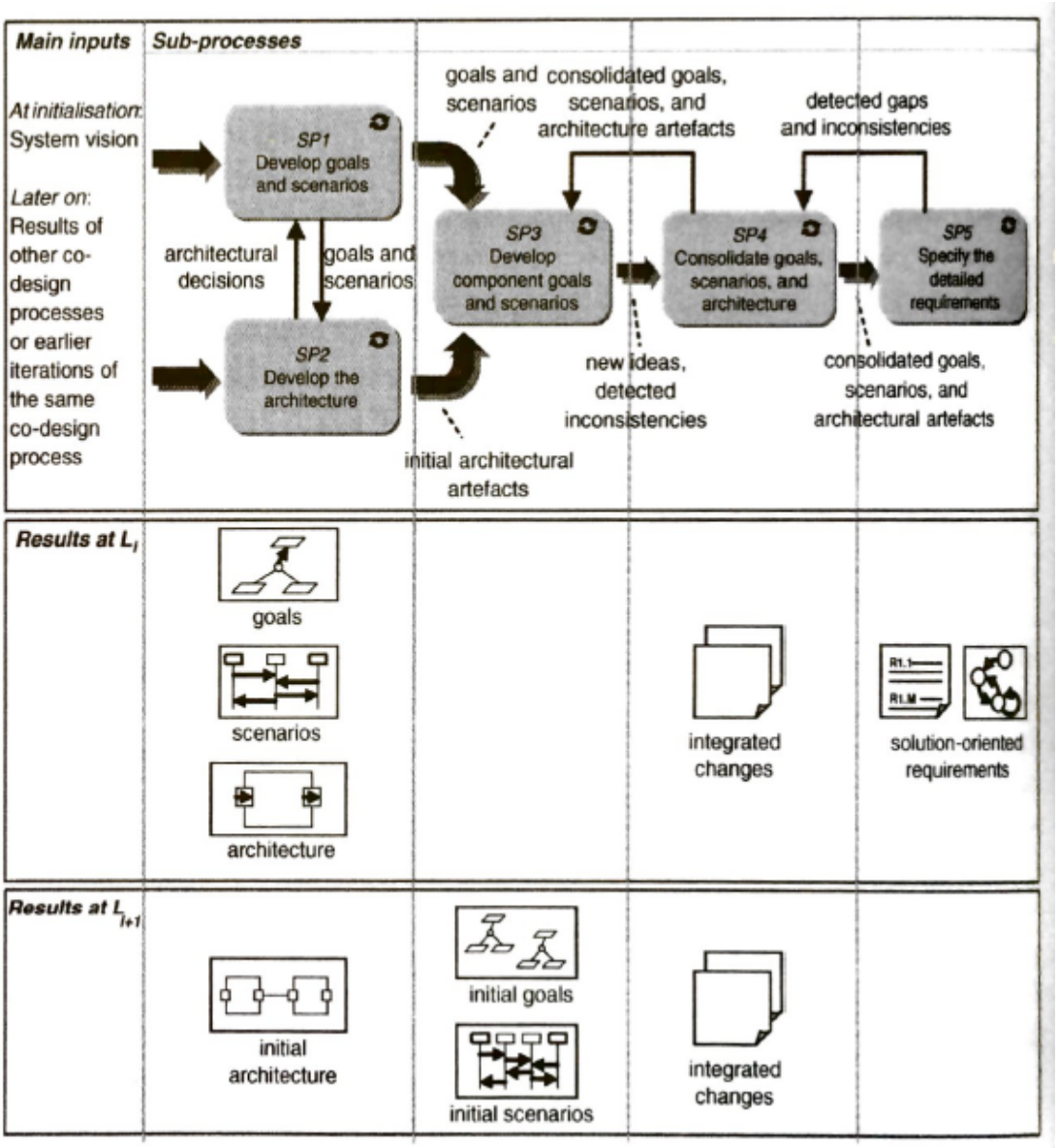

Figura 16 - Visão Geral dos cinco sub-processos [POHLKLAUS]

Sub-processo SP1: apoia o desenvolvimento de objetivos iniciais e os cenários na camada de Li.

Sub-processo SP2: apoia o desenvolvimento de artefatos de arquitetura iniciais na camada de Li+1. Sub-Processos SP1 e SP2 podem ser realizados de uma maneira entrelaçada ou, parcialmente, em paralelo.

Sub-processo SP3: os objetivos e cenários para os componentes de arquitetura na camada de Li+1 são definidos com base nos objetivos e cenários definidos na camada L1 e a arquitetura inicial definida na camada L1+1. Nesse sentido, as responsabilidades para satisfazer os objetivos e cenários na camada L1+1 são atribuídos aos componentes de arquitetura e as inconsistências entre os objetivos, cenários, e arquitetura são identificadas.

Sub-processo SP4: é responsável por conciliar os requisitos e a arquitetura das duas camadas vizinhas L1 e L1+1. Isso resulta em correções e alterações aplicadas aos artefatos de requisitos e os artefatos de arquitetura em ambas as camadas.

Sub-processo SP5: o detalhamento dos requisitos (orientado a solução) na camada L1 são especificados com base nos resultados obtidos a partir dos outros sub-processos.

Os cinco sub-processos incorporam um procedimento genérico, que pode ser aplicado em diferentes camadas de abstração, isto é, para o sistema como um todo, para componentes funcionais individuais, ou para componentes de hardware e software individuais. Em outras palavras, este procedimento é aplicável em cada processo de design.

4. PROCESSO DE ESPECIFICAÇÃO DE REQUISITOS PROPOSTO

4.1. INTRODUÇÃO

Neste capítulo é apresentado o modelo de processo proposto para produção de especificação de requisitos baseada no método COSMOD-RE. De forma a promover auxílio ao Engenheiro de Requisitos na análise, especificação e validação de requisitos, buscando aumentar a padronização da escrita e minimizar a ocorrência de inconsistências dentro dos requisitos.

Alguns conceitos serão mencionados primeiramente, devido a sua utilização na descrição, deixando claro o papel que cada um deles representa dentro do modelo de processo proposto.

Papéis:

- **Engenheiro de Requisitos**: responsável pelas atividades de descoberta, análise, especificação e validação dos requisitos e gerência de requisitos.
- **Cliente**: representa um *stakeholder* do sistema, pertencente à organização cliente e que possui autorização para atuar como fornecedor de requisitos ao projeto.

Artefatos:

- **Documento de Entrada**: documentos relacionados ao projeto antes da fase de iniciação, enviados por um cliente, por exemplo, uma proposta comercial, proposta técnica, desde que sugira uma versão inicial das características e módulos do sistema.

- **Atributos dos Requisitos**: dados recolhidos ao longo do processo de análise de requisitos, desde a especificação até a aprovação, ou posteriores mudanças. Um atributo especial é a prioridade dos requisitos, cuja definição depende de negociações com o cliente e envolvem atividades como gerência de projeto.

- **Matriz de Rastreabilidade**: documentam as relações de dependência entre requisitos. A rastreabilidade documentada vertical (obrigatória) e horizontal (opcional).

- **Glossário**: registro do vocabulário comum do projeto. É criado no início do projeto e evolui ao longo do desenvolvimento do sistema, servindo como referência a todos os envolvidos.

- **Documento de Visão**: definição da visão que os envolvidos têm do produto a ser entregue, em termos das principais necessidades e características.

- **Especificação de Requisitos de Software**: captura todos os requisitos de software do sistema e consiste em um pacote contendo especificações funcionais e não funcionais aplicáveis.

- **Lista de Requisitos**: uma lista comum que contém todos os requisitos identificados.

O modelo de processo proposto está dividido em 3 fases: Análise, Especificação e Validação. Nas próximas seções, serão apresentadas uma descrição para cada fase mostrando o que é abordado, qual o objetivo e seus artefatos de entrada e saída.

4.2. MODELO DO PROCESSO PROPOSTO

4.2.1. FASE 1: ANÁLISE

Artefatos de Entrada: Documento de entrada.

Descrição: O processo é iniciado quando Cliente e o Engenheiro de Requisitos interagem em reuniões iterativas e incrementais, debatendo as necessidades e problemas que o sistema deverá atender/solucionar. A cada nova reunião os assuntos discutidos, bem como as necessidades já identificadas, são retomados para que sejam incrementados até que um consenso final seja elaborado.

Nesse momento, o Engenheiro de Requisitos inicia o processo de abstração do sistema em camadas. Devido ao nível de informações presentes nessa fase de análise, são priorizadas as camadas de sistema e se possível a decomposição funcional, porém, se houver entendimento, o ER poderá registrar informações relacionadas as outras camadas de particionamento de hardware e software e de implantação.

Em seguida, o Engenheiro de Requisitos transcreve as necessidades passadas pelo cliente e identificadas. Busca-se que todo o processo seja desenvolvido nas reuniões com o cliente, ou seja, todas as fases do processo proposto terão seu início e fim durante as reuniões. Quando isso não for possível, o Engenheiro de Requisitos deve terminar as fases o mais rápido possível para que a validação dos requisitos possa ser feita por ele e o cliente em um momento próximo.

Objetivo: definir um ponto de vista específico do sistema, independente da tecnologia de implementação e estrutura interna que será adotada. Conforme descrito no método COSMOD-RE, o sistema é considerado uma caixa preta, tendo como foco principal a incorporação do sistema no ambiente. As interações entre o sistema e seu ambiente são definidas em termos de cenários.

Artefatos de Saída: Documento de Visão.

4.2.2. FASE 2: ESPECIFICAÇÃO

Artefatos de Entrada: Documento de Visão.

Descrição: A partir do esclarecimento das necessidades do Cliente, o Engenheiro de Requisitos conseguirá identificar os requisitos passando pelas quatro camadas de abstrações do COSMOD-RE e poderá criar até quatro tipos de artefatos (Objetivos, Cenários, Requisitos Orientados a Solução e Arquitetura).

Na camada de sistema o comportamento externamente visível é especificado através de modelos de requisitos orientado a solução, com diagramas de casos de uso. Dessa forma, os métodos de interação com o ambiente externo são definidos através das interfaces, atores e casos de uso.

São definidas unidades de funcionalidade coerentes, no qual o COSMOD-RE chama de componente funcional. Portanto, os componentes e suas conexões são definidas contribuindo para a definição mais concreta do sistema na camada de particionamento de hardware e software.

Além da definição das unidades de funcionalidades, requisitos de qualidade também serão definidos, por exemplo, adaptabilidade, reutilização, modificabilidade, entre outros. A partir dos requisitos de qualidade, os componentes de hardware e software são definidos, visando atendê-los.

Na camada de implantação do sistema, a comunicação entre esses módulos deve ser considerada, de tal forma que os requisitos definidos na camada de particionamento de hardware e software estarão agrupados para formar unidades físicas.

Para o modelo de processo proposto, um requisito estará completo e adequado quando estiver descrito em ao menos um artefato dos quatro tipos descritos pelo método COSMOD-RE, sendo prioritariamente a descrição do Requisito orientado a solução, dessa forma o detalhamento de funções, dados, comportamentos bem como atributos de qualidade estarão definidos e classificados através da utilização de modelos como fluxo de dados, de classe, ou de estado.

Nesse momento, a Matriz de Rastreabilidade é gerada de maneira que os requisitos sejam identificados através das camadas de abstração.

Também, cria-se um glossário, visando o registro do vocabulário comum ao projeto e requisitos.

Objetivo: Com esta fase, o processo proposto tem 3 objetivos.

1. Determinar a coesão dos requisitos através dos objetivos das camadas e geração dos artefatos necessários.
2. Esclarecer as ambiguidades que poderão ocorrer em um projeto quando a quantidade de requisitos é muito grande e alguns requisitos se tornam muito parecidos com outros. Serão os artefatos que esclarecerão as dúvidas que surgirem sobre a que parte do projeto pertence cada requisito em conflito.
3. Detalhamento e compreensão dos requisitos através da geração do principal artefato (Requisito orientado a solução).

Artefatos de Saída: Especificação dos Requisitos em conjunto com os Atributos dos Requisitos, Matriz de Rastreabilidade, Glossário.

4.2.3. FASE 3: VALIDAÇÃO

Artefatos de Entrada: Especificação dos Requisitos em conjunto com os Atributos dos Requisitos e Matriz de Rastreabilidade.

Descrição: Neste momento, de acordo com os níveis de rigorosidade definidos pelo Engenheiro de Requisitos e o Cliente, a Especificação dos Requisitos é avaliada e validada.

A avaliação deverá ocorrer em conjunto e esta baseada na verificação da utilização dos artefatos corretos, identificando se as necessidades esperadas estão sendo representadas pelos seus respectivos artefatos.

O parâmetro de avaliação leva em consideração os objetivos definidos na camada de sistema, ou seja, o que o sistema deverá realizar e os artefatos de maior nível de granularidade dos requisitos definidos.
Caso o cliente e o Engenheiro de Requisitos entendam a validação como negativa, propõe-se que sejam alterados os itens que contém inconsistências, e após essas alterações ocorra uma nova validação.

Uma validação negativa pode representar que a necessidade do cliente não estava corretamente traduzida na forma de requisito, fato que pode ser ocasionado pelo Cliente passar informações incorretas ou pelo Engenheiro de Requisitos não conseguir identificar corretamente a necessidade passada.

No caso de o Engenheiro de Requisitos e o Cliente entenderem a validação como positiva propõe-se que todos os requisitos sejam listados, essa lista será comum a todos e conterá todos os requisitos identificados, disponíveis e com fácil visualização.

A validação positiva representa que o cliente pôde ler, compreender e validar o requisito descrito pelo Engenheiro de Requisitos. Desta forma, tem-se a confirmação de que a especificação do requisito está em conformidade com a necessidade que o Cliente tem para com o software a ser desenvolvido.

Objetivo: Validar se os requisitos descritos refletem as necessidades passadas pelo cliente, contendo a padronização de descrição proposta e alcançando a qualidade esperada. Neste momento o Cliente pode perceber a tradução da sua necessidade expressa anteriormente em um requisito para o software. Assim a sua análise, juntamente com o Engenheiro de Requisitos, pode confirmar se o requisito criado é realmente o requisito desejado.

Artefatos de Saída: Lista de Requisitos contendo todos os requisitos especificados.

Após o término de todas as reuniões, e todos os requisitos definidos e validados será possível a criação de um modelo de Documento de Requisitos de Software utilizando os artefatos que o processo propõe. O processo não especifica um modelo ou template padrão, por não ser a intenção do trabalho, apenas busca-se oferecer todos os requisitos validados e armazenados, atribuídos a um projeto de software em desenvolvimento.

5. ESTUDO EXPERIMENTAL

5.1. DESCRIÇÃO

O estudo experimental tem como objetivo identificar características específicas relacionadas à documentação dos requisitos através de aplicação do método COSMOD-RE a fim de propor um modelo de processo que englobe tanto as características estudadas na literatura como a realidade encontrada nas empresas.

Com este propósito foi realizado em paralelo tanto o processo atualmente seguido pela organização quanto o processo proposto através do método COSMOD-RE, com intuito de obtenção de resultados comparativos obtidos ao final da fase de análise de requisitos culminando no documento de especificação de requisitos.

5.1.1. CARACTERIZAÇÃO DA ORGANIZAÇÃO

O estudo experimental foi desenvolvido em uma célula de desenvolvimento de software responsável pelos sistemas de gestão administrativos internos proprietários (ERP), composta por uma equipe de 6 colaboradores, de uma organização de grande porte.

A organização esta localizada na cidade de São Paulo, estado de São Paulo, Brasil, e possui aproximadamente 500 colaboradores, desses cerca de 200 estão diretamente relacionados a área de desenvolvimento de software. Sua principal atividade é a de serviços comunicação de transferência eletrônica de fundos.

5.1.2. CARACTERIZAÇÃO DA ESPECIFICAÇÃO DE REQUISITOS ANALISADA

Nesta pesquisa foi analisado o documento de especificação de requisitos atualmente utilizado pela organização.

Esse documento trata dos aspectos necessários para o desenvolvimento dos projetos internos e especificam as funcionalidades necessárias para o desenvolvimento do software.

As informações contidas são oriundas de documentos, e-mails e reuniões, sem nenhuma espécie de padronização, escritos por colaboradores da empresa.

De acordo com os analistas da célula participante do estudo experimental, as informações presentes nestes documentos dificilmente contemplam todas as informações necessárias para o desenvolvimento do software. Normalmente as dúvidas são esclarecidas por telefone ou diretamente com os clientes e inclusive durante a fase de codificação, ocasionando o comprometimento da qualidade dos sistemas desenvolvidos.

A imagem abaixo mostra o padrão do documento de requisitos utilizado pela organização atualmente.

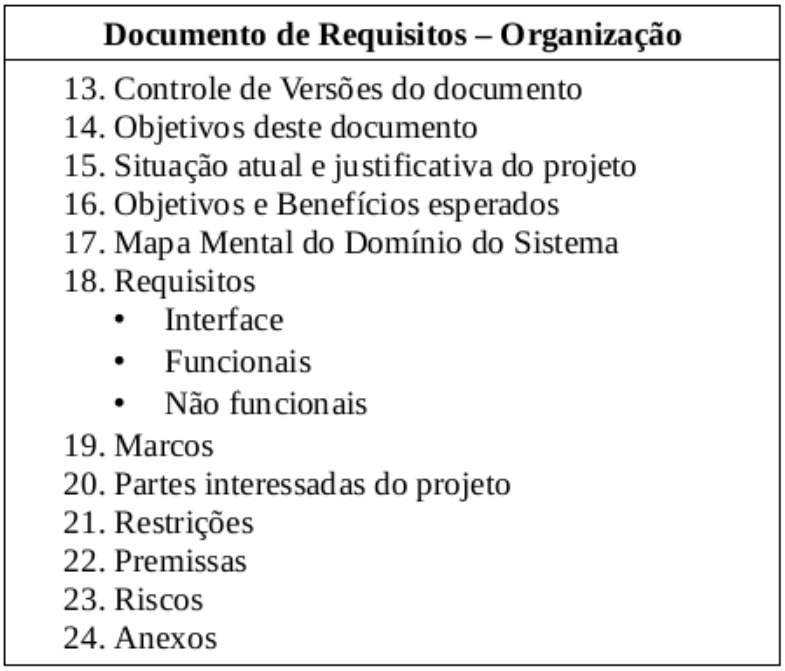

Figura 17 - SRS padrão da organização

Vale ressaltar que existem vários padrões de escrita para documento de especificação de requisitos e estes podem e devem ser adaptados conforme as necessidades, inclusive o método COSMOD-RE não sugere nenhum padrão específico e não é foco desta pesquisa padronizar uma documentação de especificação de requisitos, porém é de fundamental importância a criação de um padrão que possibilite a documentação dos requisitos buscando qualidade e atendendo as etapas do método COSMOD-RE.

5.1.3. CARACTERIZAÇÃO DO SISTEMA

O projeto, em que o método proposto foi aplicado, teve por objetivo o desenvolvimento uma aplicação para o gerenciamento dos adiantamentos de despesas, permitindo a atualização, o acompanhamento e a administração de todos os adiantamentos solicitados por funcionários da organização, através de uma única ferramenta, a ser acessada via portal interno WEB.

O prazo inicial estimado de implementação é de 10 meses.

A codificação do sistema será executada na linguagem PHP.

O projeto não tem restrições orçamentárias, tanto para o desenvolvimento, quanto para a implementação e distribuição na rede da empresa.

Um grande desafio para implantação do sistema está relacionado à gestão de mudança, uma vez que o número total de usuários diretos é de aproximadamente 220 pessoas.

Atualmente é necessário o preenchimento de planilhas Excel para acompanhamento e controle de todos os adiantamentos solicitados, aprovados, reprovados e em andamento e há uma equipe dedicada em consolidar tais planilhas e gerar material para acompanhamento dos adiantamentos pela alta direção da organização.

5.2. EXEMPLO DE USO

Durante a Fase 1 do processo proposto referente a Análise:

Foi possível observar que os requisitos identificados não explicitavam todos os objetivos do projeto apresentados pelo cliente, o que por si só poderia levar ao fracasso do projeto.

Como exemplo claro, o objetivo inicial era apenas um formulário para solicitação do adiantamento, sem considerar os controles de acessos, fluxos de aprovações e reprovações e relatórios gerenciais.

Dessa forma, seguindo o princípio do COSMOD-RE, referente ao desenvolvimento baseado em objetivos e cenários, foram realizadas perguntas e definições de cenários abstratos que auxiliaram na identificação de requisitos não levantados a princípio.

A documentação sobre a necessidade de negócio e os benefícios esperados não era suficiente para que a priorização fosse realizada, tornando ainda mais importante a participação do cliente em todas as etapas da aplicação do método.

Com relação à abstração realizada, houve consenso entre o cliente e o engenheiro de requisitos no que diz respeito à importância, ou seja, ao apresentar as camadas de abstração de sistema e decomposição funcional, o entendimento e a definição de requisitos fluiu de forma mais transparente.

O Engenheiro de Requisitos não identificou requisitos em que fosse possível realizar a estimativa com precisão.

Durante a Fase 2 do processo proposto referente a Especificação:

O engenheiro de requisitos identificou requisitos funcionais através da abstração da camada de decomposição funcional, que não estavam relacionados aos requisitos identificados pelo cliente. A partir dessa

identificação, foi criado um diagrama de componentes funcionais, que auxiliou a real necessidade e entendimento por parte do cliente.

Tais requisitos não haviam sido avaliados quanto à importância, já que não deveriam fazer parte do escopo do projeto.

Isto demonstrou a eficiência da aplicação do método proposto da abstração em camadas, seguindo involuntariamente a ordem das camadas de sistema e decomposição funcional (tendo em vista que o método COSMOD-RE não define a obrigatoriedade top-down de desenvolvimento), não apenas para a definição da importância de cada característica, mas também para a identificação dos relacionamentos e dependências entre os requisitos, e para manutenção da rastreabilidade de tais informações.

O engenheiro de requisitos e o cliente não encontraram dificuldades em executar esta etapa do método.

O solicitante do projeto não atua na área de tecnologia e encontrou facilidade com os termos e técnicas de abstração propostas, inclusive através da utilização do Glossário, proposto pelo método.

Durante a Fase 3 do processo proposto referente a Validação:

A equipe de projeto identificou os riscos da não validação associados aos requisitos funcionais, do ponto de vista da implementação. Os artefatos de casos de uso, utilizados para identificação das funcionalidades e atores do sistema foi de fundamental importância para tais identificações.

Para alguns requisitos, a equipe de projeto não sentiu a necessidade de criação de critérios de validação em função da baixa complexidade.

Foi perceptível a falta de familiaridade da equipe (nesse momento, alguns desenvolvedores participaram da reunião) com a disciplina de testes (criação de cenários de validação e casos de testes).

Também foi possível observar que algumas validações identificadas para o projeto e documentadas pela equipe durante as fases 1 e 2 não foram mencionadas quando executada a validação por funcionalidade na fase 3, ou seja, as validações identificadas inicialmente antes da aplicação do método de validação se mostraram ineficientes após a aplicação.

Ao revisar as validações originais a equipe concluiu que parte das validações inicialmente identificadas estavam incoerentes com os requisitos funcionais e/ou de negócio.

A criação do artefato com a lista de requisitos foi determinante para identificação das validações e restrição do escopo do projeto.

5.3. CONSIDERAÇÕES FINAIS DO EXEMPLO DE USO

Considerações sobre a aplicação do método. Com relação à aplicação do método, pode-se observar:

O tempo total de envolvimento do cliente e o engenheiro de requisitos no projeto foi de aproximadamente 12 horas, desde a execução da fase 1 até a obtenção da especificação de requisitos.

Este tempo foi considerado aceitável pelos envolvidos. O tempo necessário para tabulação dos dados, não foi considerado.

A equipe do projeto entendeu ser necessária uma capacitação relacionada a identificação de cenários de testes, seu controle e planejamento. A falta de experiência nesta disciplina, pode interferir de forma negativa no resultado final.

Durante a execução das etapas do método, a equipe não foi direcionada. Todas as instruções foram entregues e interpretadas pelos envolvidos.

A sequência para aplicação das etapas e a continuidade para a etapa seguinte não exigiu uma dependência, sendo considerado que grande parte

das etapas eram realizadas em conjunto, reforçando o propósito da participação do cliente ser de fundamental importância.

A aplicação da camada de abstração de decomposição funcional permitiu a identificação de requisitos funcionais que não estavam diretamente relacionados aos requisitos de negócio e, desta forma, poderiam ser desprezados. Isto também poderia indicar que os requisitos de negócio não foram devidamente identificados ou documentados.

Do ponto de vista de aplicação, o resultado obtido foi considerado satisfatório.

5.4. COMPARANDO RESULTADOS

Apresenta-se nesta seção, uma avaliação comparativa entre o modelo de processo proposto e o processo atual utilizado na organização.

O objetivo é apresentar uma comparação entre os processos para melhorar o tratamento que os requisitos recebem durante sua especificação.

Na comparação feita, foram escolhidos 5 itens que, de maneira geral, demonstram como os processos citados asseguram a qualidade dos requisitos.

Os objetivos da avaliação comparativa e os itens são listados a seguir:

1. Identificação dos Requisitos: apresenta a maneira na qual os requisitos são selecionados, identificados, por quem e como.

   Processo Atual: Cliente cria os requisitos, de forma imatura.
   Processo Proposto: Cliente e Engenheiro de Requisitos durante reuniões identificam os requisitos. A abstração em camadas auxilia o entendimento e a descoberta de novos requisitos. Cria-se o artefato de Entrada e Visão, documentando e registrando as informações pertinentes aos objetivos e necessidades do cliente.

2. Análise de Requisitos: será mostrado como Engenheiro de Requisitos trata a análise dos requisitos antes de descrever a especificação.

   Processo Atual: Não é realizada uma análise profunda dos requisitos, não são especificados/descritos em documentos, e um analista realiza a partir dos requisitos imaturos, os requisitos de sistema.
   Processo Proposto: O Engenheiro de Requisitos realiza a análise dos requisitos baseado na experiência e nos contextos, incluindo a abstração em camadas proposta pelo método COSMOD-RE. A partir da análise, modelos UML são criados com objetivo de identificar os requisitos mais detalhadamente.

3. Especificação de Requisitos: representa a maneira que os requisitos são especificados e armazenados.

   Processo Atual: A especificação vem do cliente, escrita em Linguagem Natural.
   Processo Proposto: O Engenheiro de Requisitos cria a especificação utilizando Linguagem Natural e casos de uso para demonstrar as funcionalidades do sistema. Também são utilizadas outras modelagens UML para auxiliar o processo, como diagrama de componentes funcionais definidos na camada de abstração de decomposição funcional. As interfaces e conexões entre componentes e sistemas são descobertas e alinhadas.

4. Validação de Requisitos: trata da validação de requisitos e de qual forma ela é realizada.

   Processo Atual: As validações dos requisitos são realizadas de forma subjetivas e incompletas, impossibilitando a garantia da qualidade.
   Processo Proposto: As validações dos requisitos são realizadas através do Cliente e Engenheiro de Requisitos durante reuniões e níveis de rigorosidade definidos, também pode-se solicitar a participação da equipe de desenvolvimento com intuito de garantir uma melhor validação. O artefato com a lista de requisitos é gerado.

5. Participação do Cliente: este item de comparação apresenta como um maior nível de participação do cliente dentro do processo de Engenharia de Requisitos corresponde a um maior nível de qualidade na especificação.

   Processo Atual: Baixo nível de participação, o Cliente participa através de troca de e-mails, e muitas vezes os analistas devem tomar decisões por conta própria.
   Processo Proposto: Alto, identificando e validando os requisitos através da participação constante em reuniões, troca de artefatos e apresentações.

Esta comparação entre o processo atual abordado pela organização e o processo proposto ajuda-nos a perceber que existe um grau de deficiência e incertezas tornando significativo a aplicação do método em busca pelo aumento da qualidade dos requisitos e posteriormente da solução desenvolvida.

Com este comparativo reforçam-se as ideias que serviram como base para a criação do modelo de processo apresentado. Elas dizem respeito à utilização de processos específicos para tratar os requisitos, utilizar contextos explícitos para prevenir inconsistências e diminuir redundâncias em requisitos conflitantes.

6. CONSIDERAÇÕES FINAIS

6.1. CONCLUSÃO

Este trabalho apresentou o esforço existente para que a descrição e a documentação dos requisitos estejam com maior nível de qualidade, utilizando processos e técnicas originadas do método COSMOD-RE, para que Cliente e Engenheiro de Requisitos possam ficar seguros de que o requisito criado corresponde exatamente ao requisito desejado.

Levando em consideração que o sistema final será desenvolvido a partir das atividades da Engenharia de Requisitos, esse momento do processo de desenvolvimento de software torna-se crítico.

Na Engenharia de Software, essa disciplina é relacionada como um dos principais desafios, e como a principal razão de muitas das falhas ocorridas nos sistemas desenvolvidos, já que se os requisitos não forem eficientemente descobertos, documentados, validados e gerenciados, informações podem ser perdidas e/ou desenvolvidas erroneamente.

Dentro da literatura de Engenharia de Requisitos existem diversos processos e técnicas que buscam prover auxílio e aumento da qualidade na Especificação de Requisitos.

Dessa forma, foram realizadas comparações entre alguns itens que tratam da qualidade de requisitos para reforçar que a utilização de um processo específico e definido para o entendimento, descoberta e registro de requisitos, podem ser explorados no desenvolvimento de software.

O modelo de processo proposto apresentado visa atender estes itens, provendo os benefícios citados, melhorando o processo da descoberta dos requisitos, diminuindo a quantidade de retrabalho necessário nas fases posteriores do desenvolvimento e minimizando a chance de inconsistências na descrição.

A redução de problemas na descrição e ambiguidade dos requisitos são alguns itens que o estudo experimental se mostrou possível de obtenção de melhorias, além da diminuição e melhor aproveitamento no tempo da documentação da Especificação de Requisitos.

A utilização, durante todo o processo de descoberta de requisitos, dos documentos de especificação de requisitos em conjunto com o modelo de casos de uso pôde contribuir para que o resultado alcançado ao final deste processo fosse considerado satisfatório por todos os interessados. Se estes documentos estiverem sempre alinhados entre si há uma possibilidade maior de que o processo de desenvolvimento siga sempre pelo caminho ideal, chegando ao resultado esperado.

6.2. TRABALHOS FUTUROS

Existem algumas frentes que poderão ser desenvolvidas para que ocorra a melhoria e evolução do estudo realizado até o momento como continuação do trabalho.

O desenvolvimento e a inclusão no processo de uma especificação padrão, melhorando a documentação de requisitos proposta pelo processo, bem como prover um alinhamento entre o modelo de processo e a Gerência de Requisitos, para que, além de tratados, os requisitos possam ser gerenciados de acordo com os níveis de qualidade esperados.

Olhando sobre a perspectiva ágil, o estudo experimental foi realizado em apenas um ciclo do processo de desenvolvimento de software, deixando aberto o acompanhamento do processo inteiro de desenvolvimento de software, repetindo o ciclo da Gerência de Requisitos a cada incremento.

Para a avaliação contínua do modelo de processo, a criação e o desenvolvimento de outros estudos de caso seria algo de extrema importância. Não somente pelo fato de aumentar a validação do modelo de processo, mas também para obter outros dados, em outras situações, que

poderiam influenciar determinadas fases ou passos específicos do processo, assim conseguiríamos trabalhar e evoluir estes pontos específicos.

O desenvolvimento de uma ferramenta CASE poderia ser uma outra frente de desenvolvimento importante, implementando conceitos e utilizando o auxílio computacional visando aumentar os benefícios obtidos.

Neste sentido, a continuação do trabalho deve abranger a área de Engenharia de Requisitos, utilizando o modelo de processo, buscando melhorar o que já é oferecido e preencher as lacunas deixadas.

BIBLIOGRAFIA

[CRUZ2013] CRUZ, Fabio. **Scrum e PMBOK®**: unidos no Gerenciamento de Projetos. 1ed. Rio de Janeiro: Brasport, 2013.

[HELIO2010] ENGHOLM Júnior, Hélio. **Engenharia de Software na prática**. 1ed. São Paulo: Novatec 2010.

[ISO12207] ISO 12207 - Engenharia de sistemas e software — Processos de ciclo de vida de software – Março 2009.

[KRUCHTEN2003] KRUCHTEN, Philippe. **Introdução ao RUP** – Rational Unified Process. 2ed. Rio de Janeiro: Ciência Moderna, 2003.

[LARMAN2007] LARMAN, Craig. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3ed. Porto Alegre: Bookman, 2007.

[MACH2016] MACHADO, Felipe Nery Rodrigues. **Análise e gestão de requisitos de software**: onde nascem os sistemas. 3ed. São Paulo: Érica, 2016.

[POHLKLAUS] POHL, Klaus. **Requirements Engineering**. Fundamentals, Principles, and Techniques. 1ed. Springer, 2010.

[PRES2011] PRESSMAN, Roger S. **Engenharia de Software**: uma abordagem profissional. 7ed. Porto Alegre: AMGH, 2011.

WEBGRAFIA

[FABRI2015] FABRI, José Augusto. Engenharia de Software. Disponível em: < https://engenhariasoftware.wordpress.com/>, Acesso em: 27 de julho de 2015.

[IEEE830] IEEE Std 830, http://www.urisan.tche.br/~pbetencourt/engsoftI/IEEE830/caracteristicas.html, Acesso em: 09 de outubro de 2016.

[LEITE2015] LEITE, Jair C. Engenharia de Software. Disponível em: < http://engenhariadesoftware.blogspot.com.br>, Acesso em: 27 de julho de 2015.

[CHAOS2015] The Standish Group. Chaos Report. Disponível em: < http://www.standishgroup.com>, Acesso em: 28 de setembro de 2015.

REFERÊNCIAS COMPLEMENTARES

[MELO2013] MELO, Ana Cristina de, *Repositório para o rastreamento de requisitos funcionais derivados de regras de negócio*. 2013. 115f. Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, São Paulo, 2013.

[RAFAELFIG] GRZEBIELUKA, Rafael de Figueiredo, *Qualidade de Software com Scrum*. 2015. 11f. Artigo (MBA em Gestão de Tecnologia da Informação) - Faculdade Santana, São Paulo, 2015.

[SANTANA2014] SANTANA, Gláucia Rodrigues Sirquera, *Roteiro para identificação e análise dos requisitos não funcionais em projetos SCRUM aplicando as práticas do ATAM*. 2014. 62f. Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, São Paulo, 2014.

GLOSSÁRIO

Framework	Framework conceitual ou arcabouço é usado para resolver um problema de um domínio específico, sendo que existem dois tipos: framework vertical (conhecidos como especialistas, são confeccionados através da experiência obtida em um determinado contexto específico, tentando resolver problemas de um determinado domínio de aplicação), ou framework horizontais (não dependem do domínio de aplicação e podem ser usados em diferentes domínios, ou seja, podem tentar resolver problemas de qualquer domínio de aplicação) [CRUZ2013].
Processo	É uma estrutura composta dos grupos das tarefas necessárias para a produção de software de alta qualidade. Esta estrutura define as recomendações que devem ser seguidas na produção do software [PRES2011].
Scrum	O Scrum é um framework para gerenciamento de projetos ágeis que, apesar de muito utilizado na área de desenvolvimento de software, pode ser utilizado para o planejamento, gerenciamento e desenvolvimento de qualquer produto, principalmente por ser um framework iterativo e incremental [CRUZ2013].
Taxonomia	**1** Estudo dos princípios gerais da classificação científica. **2** Distinção, ordenação e nomenclatura sistemáticas de grupos típicos, dentro de um campo científico. **4** Gram Parte que trata da classificação das palavras. Var: taxinomia e taxonomia. (http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=taxionomia acessado em 06/04/2016).