Resumo

O material a seguir é uma compilação de textos inerentes ao tema da Engenharia de Requisitos e destina-se primordialmente à qualificação de profissionais de Especialização em Dispositivos Móveis e Internet e áreas afins. Combina matérias e textos obtidos a partir de livros e de fontes abertas que devem ser creditados a seus criadores e detentores de direitos autorais. Nosso trabalho aqui foi apenas agrupá-los de maneira lógica para transmitir a ideia desejada, de modo a contribuir fortemente na formação das pessoas que dele lançarem mão para seus estudos. Os textos estão diagramados com o objetivo de articular suas argumentações para transmitir, primeiro, definições inerentes à Engenharia de Requisitos, esmiuçando conceitos como requisitos funcionais, não funcionais e de domínio, além de requisitos de usuário, sistema e software, para depois entrar nos processos de Engenharia de Requisitos, propriamente ditos, quando então, são tratados conceitos como estudos de viabilidade; levantamento e análise de requisitos; validação de requisitos; e, gerenciamento de requisitos.

Palavras-chave: Engenharia de Requisitos; Requisitos Funcionais; Requisitos Não Funcionais; Requisitos de Domínio; Requisitos de Usuário; Requisitos de Sistema; Requisitos de Software; Gerenciamento de Requisitos.

1. Engenharia de Requisitos

Em um processo de software, a engenharia de requisitos é a primeira atividade importante, após a conclusão de um relatório de necessidades resultante de um processo de pré-desenvolvimento. A engenharia de requisitos é definida em função de suas atividades principais: entendimento dos problemas (descritos em relatório de necessidades), determinação de soluções e especificação de uma solução que é testável, compreensível, manutenível e que satisfaça às diretrizes de qualidade do projeto.

Enquanto o dicionário Aurélio define ***requisito*** como “condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo fim”, o mesmo também define ***especificação*** como sendo a “descrição minuciosa das características que um material, obra, ou serviço deverão apresentar”. Dessa forma, ***requisito*** é algo diferente de ***especificação***.

Dessa forma, Engenharia de Requisitos estabelece o processo de **definição de requisitos** como aquele no qual o que deve ser feito é ***elicitado***, modelado e analisado. Este processo deve lidar com diferentes pontos de vista, e usar combinação de métodos, ferramentas e pessoal. O produto obtido é um modelo, do qual um documento de requisitos é produzido. Este processo acontece num contexto previamente definido, chamada de Universo de Informação.

Por outro lado, apenas para conceituar, ***elicitação*** é um substantivo feminino que significa a ação ou efeito de elicitar, de fazer sair, de expulsar. É uma variação do termo eliciação. É a obtenção de informações detalhadas sobre o que se pretende fazer. É o estímulo que desencadeia comportamentos típicos. Eliciar é um termo técnico oriundo da Psicologia, mais especificamente da escola Behaviorista (Comportamental). Assim, grosso modo, é o ato de provocar uma resposta ou reação em algo ou alguém. Assim, para a Engenharia de Requisitos, ***Elicitação de Requisitos*** é o levantamento e identificação de problemas, para buscar uma solução tecnológica, partindo da análise das necessidades dos usuários e do negócio.

A Engenharia de Requisitos de Software (ERS) é um processo e, como tal, deve-se ter em mente que o termo ***processo***, neste contexto, refere-se a um conjunto de atividades bem definidas com os respectivos responsáveis por sua execução, ferramentas de apoio e artefatos produzidos. Ou seja, define-se como a equipe deverá trabalhar para alcançar o objetivo, que é: desenvolver software com qualidade dentro de prazos, custos e requisitos definidos.

Assim, ERS é uma lista de tarefas e análises que geram documentação de todos os processos necessários para a produção de um programa ou sistema. Além de documentar, ela é responsável pela sua manutenção ao longo do tempo e tem etapas bem definidas para estruturar a operação. Em resumo, esse processo é dividido em **levantamento, análise, documentação, verificação, validação e garantia de qualidade e gerência**.

Mas no que diz respeito à elaboração de projetos de software, muitas vezes pode ser bastante difícil compreender a natureza dos problemas, especialmente se o sistema for novo, o que acaba dificultando a compreensão exata do que o sistema deve fazer. Neste sentido, é muito importante descrever minuciosamente as funções e restrições do sistema, de modo a descobrir, analisar, documentar e verificar o funcionamento dessas funções diante de suas restrições. Na Engenharia de Software, a parte dedicada ao estudo desses requisitos é a Engenharia de Requisitos.

A ERS é um estágio extremamente importante do processo de desenvolvimento de um software, uma vez que erros nesse estágio inevitavelmente acabam produzindo problemas posteriores no projeto e na implementação do sistema. Dessa forma, a Engenharia de Requisitos leva à produção de documentação de especificações do sistema, em que os requisitos geralmente são apresentados em dois níveis de detalhes: enquanto **usuários finais e clientes** necessitam de uma **declaração de alto nível** dos requisitos, os **desenvolvedores** de sistema precisam de uma **especificação mais detalhada do sistema**.

Todas as definições acabam chegando ao mesmo lugar, principalmente porque a Engenharia de Requisitos é bastante ampla. Mas de maneira geral, os ***requisitos*** são um

conjunto de necessidades que foram previamente explicitadas por um cliente e, em algum momento, precisarão ser cumpridas no projeto.

A Engenharia de Requisitos possui quatro fases principais:

1 – ***Estudo de viabilidade***: É feita estimativa para verificar se as necessidades dos usuários que foram identificadas podem ser satisfeitas através dos atuais recursos tecnológicos de hardware e software. Esse estudo permite decidir se o sistema proposto será viável do ponto de vista comercial, e se poderá se desenvolvido considerando as restrições orçamentárias. Um estudo de viabilidade deve ser relativamente barato e rápido e o seu resultado deve informar se a decisão é a de prosseguir ou não com uma análise mais detalhada.

2 – ***Levantamento e análise de requisitos***: é o processo de obter os requisitos do sistema através da observação de sistemas existentes; conversa com usuários e compradores em potencial; análise de tarefas; entre outras técnicas de levantamento de informações. Pode envolver o desenvolvimento de um ou mais modelos e protótipos de sistema. Tudo isso ajuda o analista a compreender o sistema a ser especificado.

3 – ***Especificação de requisitos***: compreende a tradução das informações coletadas durante a atividade de levantamento e análise em documento que defina um **conjunto de requisitos**. Dois tipos de requisitos podem ser incluídos nesse documento: os **requisitos dos usuários**, que são declarações abstratas dos requisitos de sistema para o cliente e os usuários finais do sistema; e, os **requisitos do sistema**, que compreendem uma descrição mais detalhada da funcionalidade a ser fornecida.

4 – ***Validação de requisitos***: verifica os requisitos quanto a sua pertinência, consistência e integralidade. Durante esse processo, inevitavelmente são descobertos erros de documentação de requisitos que, então, devem ser modificados a fim de corrigir esses problemas.

É sempre bom ter em mente que essas etapas não são absolutamente estanques, pois, por exemplo, a **análise de requisitos** continua durante a definição e a especificação e, mesmo, novos requisitos surgem ao longo do processo. Dessa forma, as atividades de análise, definição e especificação são intercaladas. Outro ponto a ser considerado é que o termo ***requisito*** pode assumir conotações variadas em Engenharia de Software, pois pode variar de um conceito ou declaração abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do próprio sistema, até uma definição detalhada, matematicamente formal, de uma função que o sistema deve ter. Por isso, alguns problemas que surgem durante o processo de engenharia de requisitos são resultantes da falta de uma nítida separação entre esses diferentes níveis de descrição. Os requisitos são hierarquizados, de cima para baixo, em uma pirâmide cujo topo abriga os **requisitos de negócios**. Abaixo deste vem, respectivamente, os requisitos de usuário e sistema. Para evitar confusões, pode-se utilizar o termo

requisitos de usuário para designar requisitos abstratos de alto nível; e, o termo ***requisitos de sistema***, para indicar a descrição detalhada do que o sistema deve fazer.

Em um aprimoramento dos requisitos de sistema, pode-se elaborar o ***requisito de software***, que é uma descrição dos principais recursos de um produto de software, seu fluxo de informações, comportamento e atributos. Em suma, um **requisito de software** fornece a estrutura básica para o desenvolvimento de um produto de *software*. O grau de compreensibilidade, precisão e rigor da descrição fornecida por um por um documento de requisitos de software tende a ser diretamente proporcional ao grau de qualidade do produto resultante.

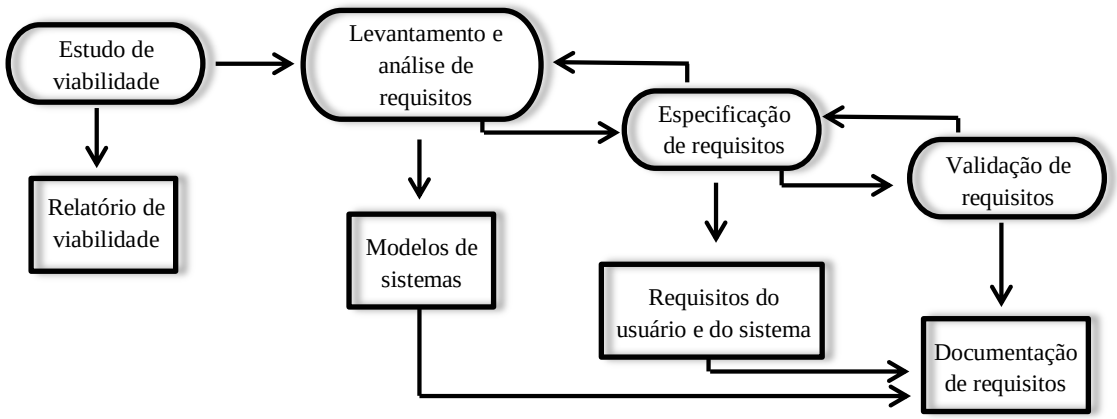
Assim, além dos requisitos de usuário e dos requisitos de sistema, uma descrição mais detalhada do projeto de software pode ser produzida para associar a engenharia de requisitos e as atividades do projeto. Os requisitos de usuário, sistema e especificações do projeto de software podem ser assim definidos:

1 – **Requisitos de usuário** são **declarações** em linguagem natural e também em diagramas, sobre funções que o sistema deve fornecer e as restrições sob as quais deve operar.

2 – **Requisitos de sistema** ou **especificação funcional** estabelecem detalhadamente as funções e as restrições de sistema e deve ser absolutamente preciso. Eventualmente, ele pode até servir como um contrato entre o comprador do sistema e o desenvolvedor do software.

3 – **Especificação do projeto de software** é uma descrição abstrata, porém detalhada, do projeto de software, que é a base para o projeto e a implementação. Essa especificação acrescenta mais detalhes à especificação de requisitos do sistema.

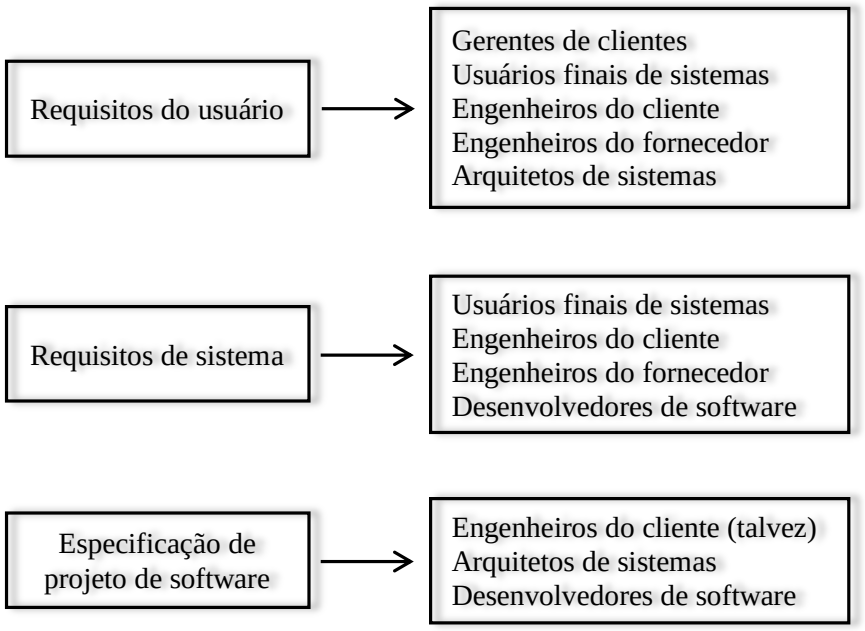
Esquematicamente, o processo de engenharia de requisitos se estrutura da seguinte maneira:



Engenharia de Requisitos

O foco principal da engenharia de requisitos é a definição e a descrição do que um sistema de software deve fazer para satisfazer aos requisitos informais fornecidos por um relatório de necessidades. O pensamento na análise de requisitos deve voltar-se principalmente **aos problemas, não às soluções**. A análise de requisitos é o principal processo executor em um sistema de feedback que produz descrições dos recursos comportamentais e não-comportamentais do software.

O uso de diferentes níveis de especificação pode ser muito útil porque se podem comunicar informações sobre o sistema para **diferentes públicos**. Por exemplo, os **requisitos de usuário** devem ser escritos para **gerentes do cliente** e dos **fornecedores** que não tenham conhecimento técnico detalhado do sistema. Já a especificação de **requisitos de sistema** deve ser voltada aos profissionais técnicos de nível sênior e gerentes de projeto, além do gerente do cliente e do fornecedor. Os usuários finais de sistemas podem ler ambos os documentos. Por fim, a especificação do projeto de software é um documento orientado à implementação e deve ser escrito para os **engenheiros de software** que desenvolverão o sistema. Dessa forma, esquematicamente, tem-se:



1.1 – Requisitos funcionais, não funcionais e de domínio

Normalmente, os **requisitos de sistema de softwares** são classificados como **funcionais, não funcionais** ou como **requisitos de domínio**:

1 – ***Requisitos funcionais (ações principais)*** são declarações de funções que o sistema deve fornecer; como o sistema deve reagir a entradas específicas; e, como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar **o que o sistema NÃO DEVE FAZER**.

2 – ***Requisitos não funcionais*** são restrições sobre os serviços ou funções oferecidos pelo sistema. Entre eles, destacam-se restrições de tempo, sobre o processo de desenvolvimento, padrões, legislação, entre outros.

3 – ***Requisitos de domínio*** são requisitos que se originam no domínio de aplicação do sistema e que refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais. Dois exemplos de requisitos do domínio são:

a) O calculo da média final de cada aluno é dado pela fórmula:

(Nota1 * 2 + Nota2 * 3)/5

b) Um aluno pode se matricular em uma disciplina desde que ele tenha sido aprovado nas disciplinas consideradas pré-requisitos.

Contudo, a distinção entre esses três tipos de requisitos (funcionais, não funcionais e de domínio) não é tão simples e clara como pode parecer. Um requisito de usuário relacionado à proteção, por exemplo, parece ser um requisito não funcional. Porém, quando desenvolvido com mais detalhes, pode levar a outros requisitos que são claramente funcionais, como a necessidade de incluir recursos de autorização de usuários no sistema.

1.1.1 – Recursos não-comportamentais da ERS

A porção não-comportamental em uma ERS identifica **e** especifica cada um dos atributos necessários do software. Esses atributos incluem ***confiabilidade, reutilizabilidade, disponibilidade, portabilidade, manutenibilidade, segurança e conformidade*** com padrões. O nível de detalhes de uma especificação não-comportamental deve ser suficiente para permitir que os projetistas e implementadores desenvolvam os componentes de um sistema que satisfaça aos requisitos e para validar produtos do processo de desenvolvimento.

A ***confiabilidade*** é um indicador de alto nível de prontidão operacional do sistema. **A chave para o aprimoramento da confiabilidade do software está na construção de históricos precisos de erros, falhas e defeitos associados às falhas do software**. Os erros, os defeitos, as falhas e as faltas têm relação de causa e efeito entre si. Um ***defeito*** é uma anomalia do produto (por exemplo, omissões ou imperfeições encontradas durante o desenvolvimento de um software). Um ***erro*** é uma ação humana

que resulta em falha no software. Uma ***falha*** é uma condição acidental que faz com que uma unidade funcional não execute a operação solicitada (por exemplo, um botão da barra de menus não responde ao clique do mouse) ou a manifestação de um erro no desenvolvimento do software.

Uma ***falha*** ocorre quando a unidade funcional interrompe a sua capacidade de executar a operação solicitada (por exemplo, quando o mostrador trava, se recusando a responder a qualquer clique do mouse). Um dos principais objetivos do processo de software eficaz é controlar os motivos dessas ***falhas***. Ao nível dos requisitos, um padrão (valores alvos) para as densidades de falhas e defeitos máximos pode ser estabelecido para o projeto de software. Por exemplo, o valor máximo aceitável para essas densidades poderia ser de quatro falhas por mil linhas de código-fonte e cinco defeitos por mil linhas de código do projeto. O propósito de fazer isso é estabelecer um padrão que as equipes de projeto possam utilizar para comparar as densidades reais com as densidades desejadas.

Para guiar o esforço da engenharia de software, podem-se fornecer os valores de destino dos requisitos não-comportamentais da qualidade de software. Isso é ilustrado em termos de requisitos de **reutilizabilidade** de um projeto de software. A ***reutilizabilidade*** é calculada considerando-se, pelo menos, quatro critérios: autodescrição (AD), modularidade (M), portabilidade (P) e independência de plataforma (IP). Dessa forma, a reutilizabilidade pode ser estimada com uma soma ponderada da seguinte forma:

$$\text{reutilizabilidade} = p_1(\text{AD}) + p_2(\text{M}) + p_3(\text{P}) + p_4(\text{IP})$$

O padrão de reutilizabilidade do projeto de software pode ser estabelecido através de um diagrama de Kiviat em “branco”, especificando os valores mínimos e máximos de cada atributo de software a ser medido. O diagrama de Kiviat oferece uma maneira fácil e conveniente de comparar as estimativas de reutilizabilidade reais durante a fase de projeto com o padrão do Projeto.

O requisito de software ***disponibilidade*** especifica os critérios necessários para garantir níveis de disponibilidade aceitáveis de um sistema. Esses critérios incluem ponto de verificação, recuperação e reiniciação. ***Ponto de verificação*** é o ponto em um programa de computador no qual o estado do programa, status ou resultados calculados são verificados ou registrados. Durante o desenvolvimento, os pontos de verificação são estabelecidos instrumentando-se o programa (inserindo comandos de escrita para observar o que o programa está fazendo em determinado estágio). A ***recuperação*** refere-se à restauração do sistema, programa, banco de dados ou outro recurso de sistema ao estado no qual o sistema possa executar as funções solicitadas. O requisito de ***portabilidade*** do software estipula os seguintes itens:

- Porcentagem dos componentes do sistema com código dependente do servidor.

- Porcentagem do código que é dependente do servidor.

- Uso de linguagem comprovadamente portável.

- Uso de compilador específico (por exemplo, Ada ou C++) ou subconjunto de linguagem.

- Uso de sistema operacional específico (por exemplo, Sun Solaris).

A ***manutenibilidade*** do sistema de software é medida de acordo com os valores médios dos critérios de consistência (c), contagem de comentários (cc), complexidade (com), modularidade (mo), tamanho (s) e grau de paralelismo (gdp). A consistência c para um único módulo de software (por exemplo, classe C++ ou Java) é medida através da contagem do número de conflitos com os requisitos (inconsistências) e calculando-se:

$$c = 1 - \frac{(\text{nº de conflitos com requisitos no módulo})}{(\text{nº de linhas de código no módulo})}$$

[consistência em único módulo]

Para uma coleção de ***n*** módulos, a consistência média é a soma das medidas das consistências do módulo individual divididas por ***n***.

$$c = \frac{\Sigma \left[1 - \frac{(\text{nº de conflitos com requisitos no módulo})}{(\text{nº de linhas de código no módulo})} \right]}{n}$$

[consistência média]

Em um único módulo, a ***complexidade*** (com) do módulo é medida pela contagem do número de pontos de decisão, mais um. O valor de ***com*** indica o número de caminhos independentes no módulo. Isso é chamado de **medida de complexidade ciclomática**. Já a ***modularidade*** (mo) é calculada comparando-se o número de módulos do sistema com o número total de variáveis ou o número de módulos com o número total de procedimentos (métodos). Logo, a ***modularidade*** é medida com a razão:

$$mo = \frac{\text{nº de módulos}}{\text{nº total de variáveis}}$$

O **grau de paralelismo** (gdp) de um único módulo de software é a contagem do número de processadores utilizados (potencial ou realmente) para obter o resultado. Nos programas Pascal, C, ou C++, o **gdp = 1**. Em um programa Java com threads (cada um sendo executado potencialmente em um processador separado), **gdp = n**. A manutenibilidade de sistemas de software com um ou mais módulos pode ser medida através da seguinte soma ponderada:

$$m = p_1c + p_2cc + (-p_3)com + p_4mo + (-p_5)s + (-p_6)gdp$$

[medida da manutenibilidade]

Os pesos na fórmula de manutenibilidade variarão e são parte da especificação dos **requisitos não-comportamentais** de um produto de software. Armazenar os valores de **com** (complexidade), **s** (tamanho) e **gdp** (grau de paralelismo) tende a diminuir a manutenibilidade. Contudo, os pesos em **com, s e gdp** são negativos. Por outro lado, aumentar os valores de **c** (consistência), **cc** (contagem de comentários) e **mo** (modularidade) tendem a aumentar a manutenibilidade do software. Por esse motivo, **c, cc e mo** têm pesos positivos. Cada medida de manutenibilidade deve ser comparada com algum parâmetro (como intervalos de medidas de manutenibilidade de projetos conhecidos, bem como um requisito mínimo de manutenibilidade para o Projeto).

Supondo-se, por exemplo, que os planejadores do projeto tenham definido o mínimo de manutenção ***m*** como 200 e que nenhum valor máximo tenha sido especificado. Além disso, considere-se que as medidas de manutenção chamadas m_2, m_3, m_5 e m_6 de quatro projetos de desenvolvimento de software similares sejam 128, 220, 190 e 55, respectivamente. Para facilitar as comparações, essas medidas podem ser colocadas em um diagrama de radar e gráfico de barras. Esses diagramas irão mostrar como a medida real ***m*** se compara com outras medidas de manutenibilidade e requisito de Projeto (***m*** solicitado). O próximo passo é medir o grau no qual a amostra do sistema de software possui os critérios utilizados para medir a manutenibilidade.

Por outro lado, a ***segurança*** de um sistema refere-se aos fatores necessários a serem utilizados para proteger o software de **acesso, utilização, modificação, distribuição acidental ou intencional, ou isenção de responsabilidade**. Os requisitos podem ser formulados levando-se em conta a necessidade de utilização de técnicas de criptografia especificadas, mantendo-se o histórico de dados, as restrições de comunicação (presença de firewalls) e verificando-se a integridade de variáveis críticas.

Padrões para requisitos são absolutamente essenciais para que se possa regular o desenvolvimento do software. Por exemplo, padrões para requisitos poderiam indicar

que a ERS seguisse a IEEE 830 para orientação de objetos ao especificar o comportamento observável do sistema, ou a ISO 9000 em preparação para a certificação de qualidade do software.

1.1.2 – Medição da qualidade dos requisitos

O modelo de sistema de feedback do processo de requisitos inclui a etapa de medição, em que a qualidade dos resultados da análise de requisitos é verificada. Pode-se medir, por exemplo, ***fidedignidade, manutenibilidade, compreensibilidade, maturidade dos requisitos, testabilidade***, entre outros.

O objetivo principal desse processo é obter uma boa especificação dos requisitos. O feedback do processo de medição é a base para as discussões entre o cliente e o especificador, para que obtenham requisitos de boa qualidade, e essa qualidade pode ser avaliada em relação a sua efetividade, utilidade e prognóstico. A ***efetividade*** de uma especificação é identificada pelo grau de solução de determinados problemas. A ***efetividade*** pode ser medida verificando-se a legibilidade, ausência de ambiguidade, consistência e integridade da especificação. Uma ***especificação*** é ***útil*** na medida em que fornece base bem definida para o projeto do software. Já a ***utilidade*** pode ser medida considerando-se a fidedignidade e compreensibilidade da especificação. O conteúdo de ***prognóstico*** da especificação de requisitos fornece um conjunto de medidas possíveis de serem utilizadas para prever a qualidade final do software.

Assim, observa-se que é possível fazer previsões relativamente precisas se forem tomadas medidas adequadas antecipadamente. Para isso, é aconselhável realizar medições dos atributos principais (como por exemplo, **manutenibilidade, portabilidade, confiabilidade**) como parte dos requisitos. As medições posteriores e projetos concluídos são fundamentais para previsões precisas de recursos em projetos futuros.

Uma lista de verificação fornece listas dos recursos que requerem especificação de requisitos, além de um método simples para a comparação de recursos planejados com recursos especificados do software. As listas de verificação são utilizadas nos casos em que não há medição de qualidade adequada para a especificação de requisitos. A medida de legibilidade é também chamada de ***índice FOG***.

1.1.3 – Requisitos de rastreabilidade bidirecional

Uma ERS é considerada ***rastreável*** se é escrita para facilitar a consulta de requisitos individuais. Além de rastrear os requisitos para os recursos de software

planejados, também é necessário que os requisitos facilitem o rastreamento de conexões entre componentes de projeto e especificações do software. Ou seja, uma ERS é rastreável se os caminhos ascendentes e descendentes entre um recurso de projeto e um requisito correspondente puderem ser identificados.

A numeração completa e cuidadosa das seções da ERS é fundamental para o rastreamento ascendente e descendente durante o processo de software. A numeração representa a primeira etapa na organização do processo de requisitos. A ***rastreabilidade*** é um atributo para se considerar a ERS de boa qualidade, incluída no IEEE 830.

Uma ERS é ***rastreada*** se o caminho ascendente puder ser identificado no requisito da ERS e nos requisitos de nível de sistema (como por exemplo, uma instrução de necessidade). Existem trabalhos que visam facilitar a criação de documentos rastreáveis com o uso de ferramentas para a Web, criando ambiente multimídia para a preparação e pesquisa de documentos com links de hipertexto.

Há diferenças entre ERS rastreável e requisito de rastreabilidade. ***Requisito de rastreabilidade*** é uma métrica de qualidade que oferece a sequência de origem a partir de uma especificação de requisitos até a seção de instrução de necessidades (***fluxo ascendente***) de uma estrutura de projeto, ou de implantação para um requisito específico (***fluxo ascendente***), ou do caminho de alocação (***fluxo descendente***) dos requisitos abaixo para a concretização do requisito.

O requisito de rastreabilidade é concluído durante a validação do sistema de software. A rastreabilidade é calculada em relação a R1 (número de requisitos atendidos por uma arquitetura ou implementação) e R2 (número de requisitos originais). Por sua vez, assume-se MT como a medida de rastreabilidade do projeto. Assim, considerando-se o número de requisitos atendidos em determinado estágio do processo de software e também a contagem do número total de requisitos originais, é possível calcular MT da seguinte forma:

$$MT = \frac{R1}{R2}$$

Assim, o documento de requisitos de software com boa qualidade fornece base compreensível e confiável para o projeto do produto e a geração de testes. Ele informa o que deve ser projetado. Ser específico em relação ao projeto significa informar aos projetistas e engenheiros de qualidade o que eles precisam saber para construir e avaliar os resultados do processo de software. Na especificação dos requisitos de software, os princípios essenciais para a ERS de qualidade são expressos nas seguintes regras:

Regra	
1. A ERS possui todos os atributos importantes	Correto, sem ambiguidade, completo, consistente, classificado (importância, estabilidade, necessidade), verificável, modificável e rastreável.
2. A ERS possui referência cruzada	Cria tabelas fazendo referência cruzada de partes da ERS e de declarações de necessidades.
3. Todos os requisitos são identificados exclusivamente	Numera seções da ERS, utiliza links de hipertexto em todos os componentes da ERS.
4. Organiza a ERS a fim de otimizar a legibilidade	Classifica os recursos, fornece índices de nomes, funções, símbolos e termos principais.

A especificação de requisitos pode ser feita de várias formas. Os principais testes para que se possa julgar a especificação de requisitos são: (1) ***legibilidade e compreensibilidade***, (2) ***fidedignidade*** e em que a extensão da especificação é concebida tendo em vista a ***verificação***, (3) ***qualidade da estrutura de verificação*** (visão do projetista da especificação) e, (4) ***qualidade da especificação*** (nível de qualidade da especificação de requisitos).

Os principais resultados da análise de requisitos são:

1 – ***Funcional (ações principais)***: a descrição funcional identifica as atividades do sistema.

2 – ***Comportamental (atividades de controle)***: a descrição comportamental descreve a sequência e a possível sobreposição das funções do sistema em uma hierarquia de atividades de controle. Essas atividades são semelhantes ao sistema nervoso central, que percebe e controla as funções do sistema em vários níveis.

3 – ***Não-comportamental (atributos)***: a distribuição não-comportamental do software inclui planejamento de engenharia humana e de garantia de qualidade.

Os principais ***produtos*** de um processo de requisitos satisfatório são:

- **Especificação de requisitos de software (ERS) completa**: é a descrição de um sistema (suas funções, seu comportamento, suas interfaces interna e externa e seus atributos de qualidade).

- **Plano de garantia de qualidade**: é a indicação da portabilidade, eficiência, confiabilidade, custos e critérios de aceitação a serem seguidos pelas equipes do projeto.

O feedback no ciclo de vida dos requisitos é proveniente da criação de protótipos, análise de riscos, simulação, elaboração de modelos e validação. **A especificação de requisitos é realmente a parte mais difícil do processo de software**, porque é altamente abstrata. Um engenheiro de requisitos vive em um mundo *anterior* à criação de qualquer coisa. O ***feedback*** dado durante o processo de requisitos faz com que a ERS se transforme em um meio prático de projetar software. A descrição do

software fornecida pela ERS apresenta a base para o processo do projeto no ciclo de desenvolvimento do software. **A engenharia de requisitos começa com a análise de problemas**.

1.1.4 – Requisitos funcionais

Muitos problemas de engenharia de software se originam justamente da imprecisão na especificação de requisitos. É natural para o desenvolvedor de sistemas interpretar requisitos ambíguos para simplificar sua implementação. Porém, muitas vezes isso NÃO é o que o cliente quer. Por isso, novos requisitos acabam tendo que ser estabelecidos e torna-se necessário realizar mudanças no sistema. Isso, naturalmente, atrasa a entrega do sistema e aumenta os custos.

Dessa forma, **requisitos funcionais devem descrever as funcionalidades ou os serviços que se esperam que o sistema forneça**. Dependem, basicamente, do tipo do software que está sendo desenvolvido, dos seus usuários e do próprio sistema. Quando expressos como requisitos de usuário, eles normalmente são descritos de forma bastante genérica, mas o conceito de requisito funcional de sistema prevê que ele descreva a função de sistema detalhadamente, suas **entradas e saídas, exceções, etc**.

Os requisitos funcionais de um sistema de software podem ser expressos de diversas maneiras e definem recursos específicos que devem ser fornecidos pelo sistema. Eles são observados no documento de requisitos de usuário de sistema e podem ser escritos em diferentes níveis de detalhes.

Em princípio, a especificação de requisitos funcionais deve ser completa e consistente, o que significa que todas as funções requeridas pelo usuário devem estar definidas e que os requisitos não devem ter definições contraditórias. Na prática, para sistemas complexos e grandes, é quase impossível atingir a consistência e o nível de completude dos requisitos. A razão disso se deve, em parte, à complexidade inerente aos sistemas e, porque diferentes pontos de vista apresentam necessidades inconsistentes que podem não ser tão obvias quando se especificam os requisitos pela primeira vez, já que normalmente os problemas emergem depois de análises mais profundas. Somente durante as revisões ou em fases posteriores do ciclo de vida, é que os problemas começam a ser descobertos e são corrigidos no documento de requisitos.

1.1.5 – Requisitos não funcionais

Como o próprio nome sugere, requisitos não funcionais são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema. Eles podem estar relacionados a propriedades de sistema emergentes, como **confiabilidade, tempo**

de resposta e espaço ocupado em disco. Alternativamente, **eles podem definir restrições para o sistema**, como a capacidade dos dispositivos de entrada e saída e as representações de dados utilizadas nas interfaces de sistema.

Muitos requisitos não funcionais dizem respeito ao sistema como um todo, e não a características individuais do mesmo. Isso significa que **eles são, frequentemente, mais importantes que os requisitos funcionais**. Enquanto a falha em cumprir com um requisito funcional individual pode degradar o sistema, a falha em cumprir um requisito não funcional pode tornar todo o sistema inútil. Contudo, requisitos não funcionais nem sempre dizem respeito ao sistema de software a ser desenvolvido. Alguns deles podem simplesmente restringir o processo que pode ser utilizado para desenvolver o sistema.

Requisitos não funcionais surgem conforme a necessidade dos usuários e em razão de restrições orçamentárias, de políticas organizacionais, pela necessidade de interoperabilidade com outros sistemas de software ou hardware ou devido a fatores externos, como por exemplo, regulamentos de segurança e legislação sobre privacidade. Normalmente, eles podem ser classificados em:

1 – ***Requisitos de produtos***: que especificam o comportamento do produto, como desempenho relacionado a velocidade de operação; espaço de memória exigido; confiabilidade; taxa aceitável de falhas; portabilidade; usabilidade, e etc.

2 – ***Requisitos organizacionais***: oriundos de políticas e procedimentos nas organizações do cliente e do desenvolvedor, tais como padrões de processos, cultura organizacional, linguagem de programação, entre outros.

3 – ***Requisitos externos***: abrangem todos os requisitos procedentes de fatores externos ao sistema e a seu processo de desenvolvimento, como interoperabilidade, requisitos legais e regras de Compliance, entre outros.

Um problema comum aos requisitos não funcionais é que eles podem ser, às vezes, de difícil verificação. Por exemplo, eles podem ser escritos para refletir os objetivos gerais do cliente, como usabilidade, habilidade do sistema se recuperar de falhas ou a velocidade de resposta ao usuário. Esses requisitos podem causar problemas sérios aos desenvolvedores, visto que deixam o enfoque aberto a discussões e interpretações. O ideal é que os requisitos não funcionais sejam expressos quantitativamente, utilizando-se métricas que possam ser objetivamente testadas.

Porém, na prática, a especificação quantitativa de requisitos é difícil. Por exemplo, os clientes podem não ser capazes de traduzir suas metas em requisitos quantitativos, pois para algumas metas, como facilidade de manutenção, não há métricas que possam ser utilizadas. Isso significa que o custo de verificar objetivamente requisitos quantitativos não funcionais pode ser muito alto. Dessa forma, os documentos de requisitos, muitas vezes, incluem declarações de metas misturadas com requisitos.

As metas podem até ser úteis aos desenvolvedores, pois fornecem pistas sobre as prioridades dos clientes. Porém, os clientes devem estar cientes de que essas metas estão sujeitas a más interpretações que não podem ser objetivamente verificadas.

Outra característica dos **requisitos não funcionais** é que eles frequentemente **entram em conflito e interagem** com **requisitos funcionais** do sistema. Em princípio, os **requisitos funcionais e não funcionais** devem ser **diferenciados em documentos de requisitos**, mas no mundo real isso é muito difícil. Se os requisitos **não funcionais** forem definidos separadamente dos **funcionais**, pode se tornar bastante difícil observar o relacionamento entre eles. Por outro lado, se eles forem definidos junto com os requisitos funcionais, poderá ser bastante difícil separar considerações funcionais e não funcionais, e identificar os requisitos que correspondem ao sistema como um todo.

Dessa forma, é preciso encontrar um equilíbrio adequado, e isso depende do tipo de sistema que está sendo especificado. Contudo, requisitos claramente relacionados às propriedades emergentes do sistema devem ser explicitamente destacados. Isso pode ser feito inserindo-os em uma seção separada no documento de requisitos ou distinguindo-os, de alguma forma, dos outros requisitos do sistema.

1.1.6 – Requisitos de domínio

Requisitos de domínio são aqueles derivados do domínio da aplicação, em vez de serem obtidos a partir das necessidades específicas dos usuários do sistema. Eles podem ser novos requisitos funcionais entre si; podem restringir requisitos funcionais existentes; ou, podem estabelecer como devem ser realizados cálculos específicos. **Os requisitos de domínio são importantes porque, muitas vezes, refletem fundamentos do domínio da aplicação**. Se esses requisitos não forem atendidos, poderá ser impossível fazer o sistema operar satisfatoriamente.

Outra coisa que deve ser levada em conta é que os **requisitos de domínio são expressos com o uso de uma linguagem que é específica do domínio da aplicação** e, muitas vezes, é difícil aos engenheiros de software compreender essa linguagem. Por outro lado, os especialistas de um domínio podem deixar de fornecer informações em determinado requisito, simplesmente porque para eles essas informações são muito óbvias. Contudo, elas podem não ser tão óbvia para os desenvolvedores de sistemas, que, como resultado disso, talvez acabem implementando o requisito de maneira insatisfatória.

1.2 – Requisitos de usuário

Os requisitos de usuários **devem descrever os requisitos funcionais e os não funcionais** de modo compreensível pelos usuários do sistema que não tenham conhecimentos técnicos detalhados. **Eles devem especificar somente o comportamento externo do sistema, evitando, tanto quanto possível, as características do projeto de sistema**. Dessa forma, não devem ser definidos utilizando-se um modelo de implementação. Eles podem ser escritos com o uso de linguagem natural, formulários e diagramas intuitivos simples.

Contudo, vários problemas podem surgir quando os requisitos são escritos em linguagem natural, tais como:

1 – ***Falta de clareza***: às vezes é difícil utilizar a linguagem de maneira precisa e sem ambiguidades, sem produzir um documento de difícil leitura.

2 – ***Confusão de requisitos***: os requisitos funcionais e os não funcionais, além dos objetivos do sistema e as informações sobre o projeto podem não estar claramente definidos.

3 – ***Fusão de requisitos***: vários requisitos diferentes podem acabar sendo expressos juntos, como um único requisito.

Assim, é boa prática separar os requisitos de usuário dos requisitos mais detalhados do sistema, em um documento de requisitos. Caso contrário, leitores que não têm conhecimentos técnicos dos requisitos de usuário podem ser sobrecarregados com detalhes que, na verdade, somente são relevantes para os técnicos. Por outro lado, quando os requisitos de usuário apresentam muitas informações, isso restringe a liberdade do desenvolvedor do sistema para oferecer soluções inovadoras aos problemas do usuário e faz com que os requisitos sejam de difícil compreensão. **Os requisitos do usuário devem simplesmente mostrar os recursos principais a serem fornecidos**.

A lógica associada com os requisitos é importante porque ela ajuda os desenvolvedores de sistemas e os responsáveis pela sua manutenção a compreender por que os requisitos foram incluídos, e a avaliar o impacto da mudança nos requisitos. Para mitigar possíveis divergências quando estiver redigindo os requisitos de usuário, é aconselhável que se sigam algumas diretrizes simples:

1 – Criar um formato-padrão e certificar-se de que todas as definições de requisitos estão de acordo com o formato. Neste caso, padronizar o formato significa que as omissões podem ser menos frequentes e faz com que os requisitos sejam verificados com mais facilidade. Busque “reforçar” o requisito inicial, considerando uma declaração de lógica, com cada requisito de usuário, e uma referência à especificação mais detalhada de requisitos de sistema;

2 – Utilizar a linguagem de modo consistente. Em particular, deve-se fazer distinção entre os requisitos obrigatórios e os que são apenas desejáveis. É prática usual definir os requisitos obrigatórios utilizando o verbo "DEVE" e os requisitos desejáveis, utiliza-se o verbo "DEVERIA". Dessa forma, é **obrigatório** que o sistema inclua um recurso para acrescentar nós em um desenho. É desejável, ainda, que a sequência de ações seja feita como foi especificado, mas isso não será absolutamente essencial, se houver boas razões para fazê-lo de outra maneira;

3 – Deve-se utilizar no texto **(negrito ou itálico**) para ressaltar partes importantes dos requisitos; e,

4 – Evitar, tanto quanto possível, o uso de jargões de informática. Contudo, inevitavelmente, ocorrerá que termos detalhados, utilizados no domínio da aplicação do sistema, sejam incluídos nos requisitos do usuário.

1.3 – Requisitos de sistema

Requisitos de sistema são descrições mais detalhadas que os requisitos do usuário. Eles podem servir como base para um contrato destinado à implantação do sistema e, portanto, devem ser uma especificação completa e consistente de todo o sistema. Eles são utilizados pelos engenheiros de software como ponto de partida para o projeto de sistema. Neste sentido, a especificação de requisitos de sistema pode incluir diferentes modelos do sistema, como um modelo de objeto ou um modelo de fluxo de dados.

Em princípio, **os requisitos de sistema deveriam definir o que o sistema teria que fazer, e não como ele poderia ser interpretado**. Porém, no que se refere aos detalhes exigidos para especificar o sistema completamente, é quase impossível excluir todas as informações de projeto. Algumas razões para isso são:

1 – Uma arquitetura inicial do sistema pode ser definida para ajudar a estruturar a especificação de requisitos. Os requisitos de sistema são organizados de acordo com os diferentes subsistemas que constituem o sistema.

2 – Na maioria dos casos, os sistemas devem interoperar com os outros sistemas existentes. Isso restringe o projeto, e essas restrições geram requisitos para o novo sistema.

3 – O uso de um projeto específico (como a programação de N-versões para atingir a confiabilidade) pode ser um requisito externo de sistema.

A linguagem natural é frequentemente utilizada para escrever especificações de requisitos de sistema. Porém, muitos outros problemas com a linguagem natural podem surgir quando ela é utilizada para especificações mais detalhadas, tais como:

1 – A compreensão da linguagem natural depende do uso das mesmas palavras para o mesmo conceito, pelos leitores e por quem escreve as especificações. Isso leva a divergências, devido à ambiguidade da linguagem natural.

2 – Uma especificação de requisitos em linguagem natural é muito flexível. Pode-se dizer a mesma coisa de modos completamente diferentes. Neste caso, fica por conta do leitor descobrir quando os requisitos são os mesmos e quando são diferentes. Lembre-se: palavras escritas não têm entonação.

3 – Não existe nenhum meio fácil de padronizar os requisitos de linguagem natural, e pode ser muito difícil encontrar todos os requisitos relacionados. Para descobrir as consequências de uma mudança, talvez seja preciso examinar cada requisito, em vez de simplesmente um grupo de requisitos relacionados.

Devido a esses problemas, as especificações de requisitos escritas em linguagem natural estão sujeitas a divergências. Muitas vezes, elas somente são descobertas em fases posteriores do processo de software e, então, a solução para elas pode ser muito dispendiosa. Por isso mesmo, existem várias alternativas para o uso da linguagem natural que adicionam estrutura á especificação e que ajudam a reduzir a ambiguidade. Por exemplo:

Notação	Descrição
Linguagem natural estruturada	Essa abordagem depende da definição de formulários-padrão ou *templates* para expressar a especificação de requisitos.
Linguagem de descrição de projeto	Essa abordagem utiliza uma linguagem, como por exemplo, linguagem de programação, mas com recursos mais abstratos para especificar os requisitos pela definição de um modelo operacional do sistema.
Notações gráficas	Uma linguagem gráfica, complementada com anotações de texto, é utilizada para definir os requisitos funcionais do sistema.
Especificações matemáticas	São anotações com base em conceitos matemáticos, como uma máquina de estados finitos e conjuntos. Essas especificações não ambíguas reduzem as discussões entre cliente e fornecedor dobre a funcionalidade do sistema. Contudo, a maioria dos clientes não compreende as especificações formais e reluta em aceita-las no momento de uma contratação de sistema.

1.3.1 – Especificações em linguagem estruturada

A linguagem natural estruturada é uma forma **restrita** da linguagem natural, que se destina a escrever requisitos de sistema. A vantagem dessa abordagem é que ela

mantém a maior parte da facilidade de expressão e compreensão da linguagem natural, mas também garante que algum grau de uniformidade seja imposto à especificação. As notações de linguagem estruturada podem limitar a terminologia utilizada e usar *templates* para especificar os requisitos de sistema. Elas também podem incorporar princípios de controle procedentes de linguagens de programação e destaques gráficos para dividir a especificação.

Para utilizar uma abordagem com base em formulários para especificar os requisitos de sistema, é preciso definir um ou mais formulários ou *templates-padrão* para expressar os requisitos. A especificação pode ser estruturada em torno dos objetos manipulados pelo sistema, das funções realizadas ou dos eventos por ele processados. Quando um formulário-padrão é utilizado para especificar os requisitos funcionais, as seguintes informações devem ser incluídas:

1 – Descrição da função ou entidade que está sendo especificada;

2 – Descrição de suas entradas e de onde elas se originam;

3 – Descrição de suas saídas e para onde elas prosseguirão;

4 – Indicação de que outras entidades são utilizadas (a parte referente a "requer");

5 – Se uma abordagem funcional for utilizada, será "chamada" uma pré-condição estabelecendo o que deve ser verdadeiro antes da função; e, também será "chamada" uma pós-condição especificando o que é verdadeiro depois da função; e,

6 – Descrição dos efeitos colaterais (se existirem) da operação.

Utilizar especificações formatadas elimina problemas de especificação em linguagem natural, uma vez que há menos variação na especificação e os requisitos são agrupados de modo mais eficaz. Contudo, pode permanecer alguma ambiguidade na especificação. Assim, métodos alternativos, que utilizam notações mais estruturadas, como Program Description Language (PDL) avançam um pouco mais em direção à resolução dos problemas de ambiguidade da especificação, mas quem não é especialista geralmente a considera de difícil compreensão.

1.3.2 – Especificação de requisitos com o uso de PDL

Ao contrário das ambiguidades inerentes à especificação em linguagem natural, com o uso de uma PDL (Linguagem de Descrição de Programa) é possível descrever os requisitos operacionalmente, com bem menos riscos de falta de compreensão do conteúdo trabalhado. PDL é uma linguagem derivada de linguagem de programação

Java ou Ada, que pode conter princípios mais abstratos, adicionais, para aumentar seu poder de expressão. A vantagem de se utilizar PDL é que ela pode ser verificada sintática ou semanticamente por ferramentas de software. Omissões e inconsistências de requisitos podem ser inferidas a partir dos resultados dessas verificações.

As PDLs resultam em especificações bastante detalhadas e, algumas vezes, estão muito perto da implementação para sua inclusão em um documento de requisitos. Mas sua aplicabilidade é melhor nos seguintes casos:

1 – Quando a operação é especificada como consequência de ações mais simples e a ordem de execução é fator importante. As descrições dessas sequências em linguagem natural são, muitas vezes, confusas, particularmente se condicionais aninhadas e loops estiverem envolvidos; e,

2 – Quando interfaces de hardware e software tiverem de ser especificadas. Em muitos casos, as interfaces entre subsistemas são definidas na especificação de requisitos de sistema. O uso de PDLs permite que tipos e objetos de interface sejam especificados.

Se o leitor dos requisitos de sistema estiver familiarizado sobre com a PDL utilizada, especificar os requisitos dessa maneira pode torna-los menos ambíguos e mais fáceis de serem compreendidos. Se a PDL tiver como base a linguagem de implementação, existirá uma natural transição dos requisitos para o projeto. A possibilidade de má interpretação fica reduzida e os especificadores não precisam ser treinados em outra linguagem descritiva. Mas também há desvantagens nesta abordagem de especificação de requisitos:

1 – A linguagem utilizada para escrever a especificação pode não ser suficientemente expressiva para descrever a funcionalidade do sistema;

2 – A notação só é compreensível para pessoas que tenham algum conhecimento de linguagem de programação; e,

3 – O requisito pode ser considerado um esboço de especificação de projeto, em vez de um modelo para ajudar o usuário a compreender o sistema.

Maneira mais efetiva de utilizar essa abordagem de especificação é combiná-la com o uso da linguagem natural estruturada. Se, por um lado, a abordagem com base em formulários pode ser utilizada para especificar o sistema completo, por outro, a PDL pode ser utilizada para definir sequências de controle ou interfaces mais detalhadamente.

1.3.3 – Análise de problemas

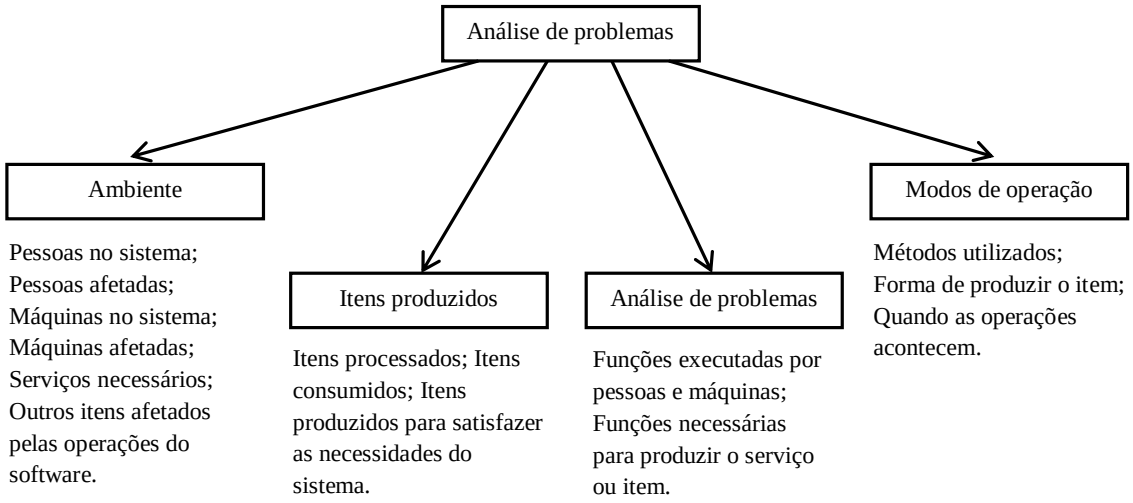
Também conhecido como ***análise de requisitos***, a análise de problemas define o espaço do produto de um processo de software, isto é, define o contexto para possíveis soluções de software para um problema. Ela resulta na identificação dos itens a serem produzidos, das principais funções executadas pelas pessoas e máquinas empregadas para produzir o produto desejado, dos métodos necessários, do cronograma de operações e, principalmente, do ambiente (pessoas afetadas por um produto de software, máquinas utilizadas ou afetadas pelo software, serviços prestados e outros itens, tais como fluxo de tráfego ou hora da comunicação). Este processo implica em particionamento, abstração e projeção.

- ***Particionamento***: agrega as relações estruturais entre os objetos, as funções e os estados, e simplifica (compartimenta) as estruturas a serem analisadas.

- ***Abstração***: identifica as relações estruturais "genérico/específico" entre objetos, funções e estados.

- ***Projeção***: fornece uma "visão" das relações estruturais entre objetos, funções ou estados. As perspectivas organizacionais dos objetos, das funções e dos estados são muito úteis no desenvolvimento da compreensão de um problema e sua solução.

A análise de problemas / requisitos fornece um ponto de partida necessário para o desenvolvimento das especificações de requisitos de software. A especificação de requisitos tem como objetivo principal descrever os objetos, as funções e os estados relacionados ao problema.



1.3.4 – Especificação de interface

A grande maioria dos sistemas deve operar com outros sistemas que já foram implementados e instalados em um ambiente. Se o novo sistema e os sistemas existentes devem trabalhar em conjunto, interfaces de sistemas existentes precisam ser especificadas com precisão. Essas especificações devem ser definidas no início do processo e incluídas (talvez como apêndice) no documento de requisitos. Existem três diferentes tipos de interfaces que podem precisar ser definidas:

1 – Interfaces de procedimento, em que subsistemas existentes oferecem uma gama de funções que são acessadas ao chamar procedimentos de interface;

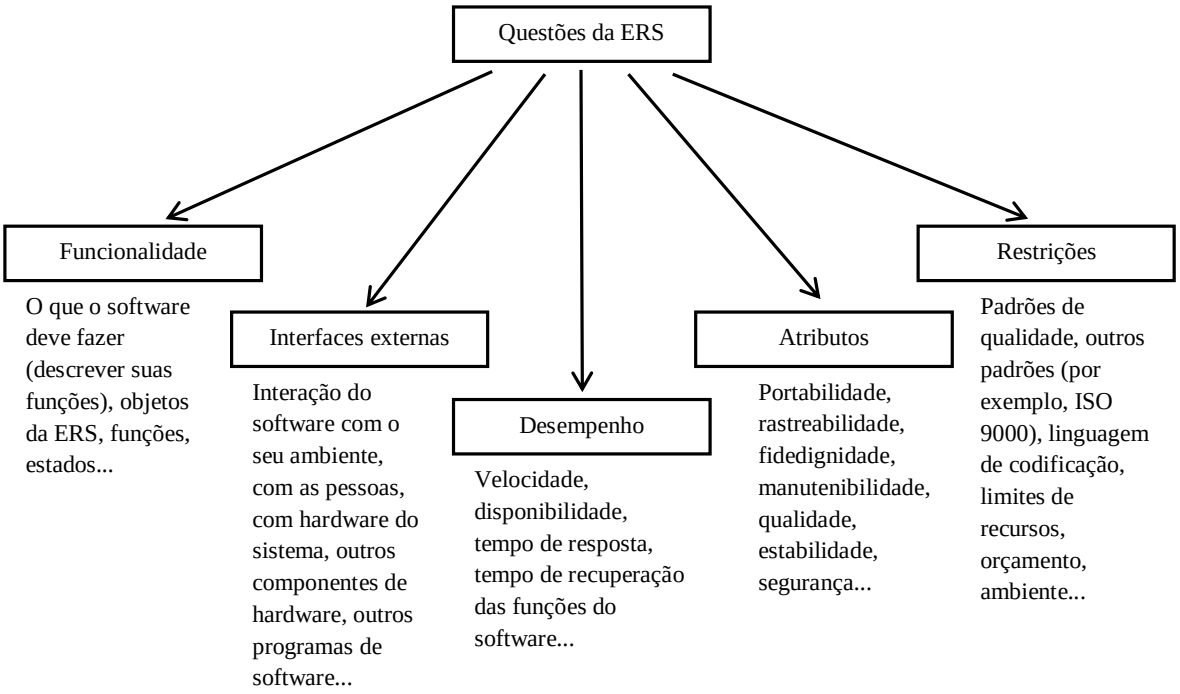
2 – Estruturas de dados, que são transmitidas de um subsistema para outro. Uma PDL baseada em Java pode ser utilizada para isso, com a estrutura de dados sendo descrita com o uso de uma definição de classe com atributos representando campos da estrutura. Contudo, os diagramas de relacionamento de entidades parecem ser melhores para esse tipo de descrição; e,

3 – Representações de dados (como a ordenação em bits) que foram estabelecidas para um subsistema existente. Contudo, o Java não é compatível com essa especificação de representação detalhada.

As notações formais permitem que as interfaces sejam definidas de maneira NÃO ambígua, mas sua natureza especializada significa que elas não são compreensíveis sem treinamento especial. Elas raramente são utilizadas na prática para a especificação de interfaces, embora sejam perfeitamente adequadas para esse propósito. Menos formais, as descrições de interfaces em PDL são uma conciliação entre a facilidade de compreensão e a precisão, mas geralmente elas são mais precisas que a especificação de interfaces em linguagem natural.

1.3.5 – Especificação de Requisitos de Software (ERS)

A ERS é a descrição de um produto de software, programa ou conjunto específico de programas que executa uma série de funções no ambiente de destino (Padrão IEEE 830-1993). A ERS emerge dos componentes da análise de problemas. Os primeiros esboços das seções de uma ERS geralmente são escritos durante o processo de decomposição resultante da análise de problemas. Ao escrever a ERS, um engenheiro de requisitos lida com cinco questões básicas:



A ERS padrão IEEE é composta de: Introdução, Descrição global, Requisitos específicos, Rastreabilidade de requisitos e Apêndices. A introdução identifica o objetivo, o escopo, as definições, os acrônimos, as abreviaturas, as referências e a visão global do documento de requisitos, onde objetivo e escopo devem estar vinculados ao relatório de necessidades do processo de desenvolvimento e fornecerão refinamentos derivados da análise de problemas. A descrição global indica os fatores gerais que influenciam os produtos (resultado de um processo de software) e seus requisitos. Ela trata da perspectiva de como o produto está relacionado a um sistema maior; funções do produto; características do usuário; restrições; hipóteses e dependências.

Por sua vez, a seção de Requisitos específicos fornece descrição do comportamento observável de um sistema de software. Ela incluir também descrição dos recursos não-comportamentais do software (desempenho, restrições de projeto e atributos de software). O comportamento observável é descrito em função de todas as entradas e saídas geradas pelas funções específicas do software. As relações entre as entradas e saídas são fornecidas. Todas as interfaces entre o software e o ambiente também são especificadas.

1.4 – O documento de requisitos de software

O **documento de requisitos de software** – às vezes, chamado de SRS (***software requirements specification***) ou **especificação e requisitos de software** – é a declaração oficial do que é exigido dos desenvolvedores de sistema. Ele deve incluir **os requisitos de usuário para o sistema e a especificação detalhada dos requisitos de sistema**. Em alguns casos, os requisitos de usuário e de sistema podem ser integrados em uma única descrição. Em outros casos, os requisitos de usuário são definidos em uma introdução à especificação dos requisitos de sistema. Se houver grande número de requisitos, os requisitos detalhados de sistema poderão ser apresentados como documentos separados.

O documento de requisitos tem um conjunto diversificado de usuários, abrangendo desde a Administração Superior da organização, que está pagando pelo sistema, até os engenheiros responsáveis pelo desenvolvimento do software. Boas práticas sugerem que existem seis requisitos aos quais o documento de requisitos de software deveria satisfazer:

- Especificar somente o comportamento externo do sistema.

- Especificar as restrições à implementação.

- Ser de fácil modificação.

- Servir como ferramenta de referência para os responsáveis pela manutenção do sistema.

- Registrar a estratégia sobre o ciclo de vida do sistema.

- Caracterizar respostas aceitáveis para eventos indesejáveis.

Contudo, algumas vezes é difícil especificar sistemas em termos do que eles farão (seu comportamento externo). Inevitavelmente, devido às restrições originárias de sistemas existentes, o projeto de sistema é restringido, e isso deve ser refletido no documento de requisito. Por outro lado, a necessidade de registrar uma estratégia sobre o ciclo de vida do sistema é algo muito bom e aceito, mas não amplamente seguido, quando são escritos os documentos de requisitos.

O padrão para documentos de requisitos mais amplamente conhecido é o padrão IEEE/ANSI 830-1993, o qual sugere a seguinte estrutura para os documentos de requisitos:

1. **Introdução**

    1.1 Propósito do documento de requisitos

    1.2 Escopo do produto

    1.3 Definições, acrônimos e abreviações

    1.4 Referências

    1.5 Visão geral do restante do documento

2. **Descrição geral**

    2.1 Perspectiva do produto

    2.2 Funções do produto

    2.3 Características do usuário

    2.4 Restrições gerais

    2.5 Suposições e dependências

3. **Requisitos específicos** que abrangem os requisitos funcionais, não funcionais e de interface. Essa é, obviamente, a parte mais substancial do documento, mas, devido à ampla variabilidade na prática organizacional, não é apropriado definir uma estrutura-padrão para essa seção. Os requisitos podem documentar interfaces externas, descrever funcionalidade e desempenho de sistema, especificar requisitos lógicos de banco de dados, restrições de projeto, propriedades emergentes do sistema e características de qualidade.

4. **Apêndices**

5. **Índice**

Embora o padrão IEEE não seja ideal, ele contém grande quantidade de boas orientações sobre como escrever os requisitos e como evitar problemas. Ele é muito geral para ser considerado um padrão organizacional por si só. Contudo, ele pode ser adaptado para definir um padrão dirigido às necessidades da organização em particular.

Naturalmente, as informações incluídas no documento de requisitos devem depender do tipo de software que está sendo desenvolvido e da abordagem de desenvolvimento utilizada. Se, por exemplo, for adotada abordagem evolucionária para um produto de software, o documento de requisitos se omitirá em relação a muitos itens sugeridos pelo padrão IEEE. Nesse caso, os projetistas e programadores devem utilizar seu julgamento para decidir como atender aos requisitos dos usuários do sistema.

Por outro lado, quando o software é parte de um grande projeto de engenharia de sistemas, que inclui sistemas de hardware e software que interagem entre si, frequentemente, é essencial definir os requisitos com um refinado nível de detalhes. Isso significa que os documentos de requisitos provavelmente serão muito extensos e, para tanto, é particularmente importante incluir tabela abrangente de conteúdo e índice de documentos, de modo que os leitores possam encontrar mais facilmente as informações desejadas.

Resumindo, os requisitos para sistemas de software estabelecem o que o sistema deve fazer e definem restrições sobre sua operação e implementação. Neste contexto, os requisitos funcionais são declarações de funções que o sistema deve fornecer, ou são descrições de como alguns cálculos devem ser realizados. Os requisitos de domínio são requisitos funcionais, provenientes de características do domínio de aplicação. Por sua vez, os requisitos não funcionais são requisitos do produto que restringem o sistema a ser desenvolvido, os requisitos de processo que se aplicam ao processo de desenvolvimento e os requisitos externos. Eles frequentemente se relacionam às propriedades emergentes do sistema e, portanto, se aplicam ao sistema como um todo.

Continuando, os requisitos de usuário se destinam às pessoas envolvidas no uso e na aquisição do sistema. Eles devem ser escritos utilizando-se linguagem natural, tabelas e diagramas, de modo que sejam compreensíveis. Já os requisitos de sistema se destinam a comunicar, de modo preciso, as funções que o sistema tem de fornecer. Para reduzir a ambiguidade, eles podem ser escritos em linguagem estruturada de algum tipo, que pode ser um formulário estruturado de linguagem natural, uma linguagem com base em linguagem de programação de alto nível ou linguagem especial para a especificação de requisitos.

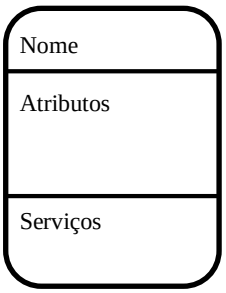
Finalmente, **o documento de requisitos de software é a declaração estabelecida dos requisitos do sistema**. Ele deve ser organizado de modo a que possa ser utilizado pelos clientes de sistema e pelos desenvolvedores de software.

1.4.1 – Análise de requisitos orientada a objetos

A abordagem orientada a objetos para a especificação de um sistema de software é caracterizada por hierarquias de objetos. Essas hierarquias começam representando um problema com um objeto contextual, que pode ser decomposto em objetos que representam “explicações” do problema em termos de subproblemas. A ideia básica é dar uma abordagem incremental à compreensão do problema, para ocultar detalhes desnecessários e facilitar a captura dos recursos essenciais de um sistema complexo. O padrão fornecido pelo IEEE inclui modelo de especificação OO.

A Análise de Requisitos Orientada a Objetos (OORA) começa com a definição dos objetos identificados em uma partição. Cinco atividades são comuns na abordagem OO da análise de problemas: especificação dos objetos, atributos, estruturas, serviços e assuntos.

Na primeira etapa da OORA, são identificadas as características de cada objeto em uma partição. São escolhidos nomes, possíveis atributos (armazenamento) e serviços para cada objeto. Os objetos são simbolizados por retângulos de cantos arredondados com três regiões horizontais: nome, lista de atributos e lista de serviços.

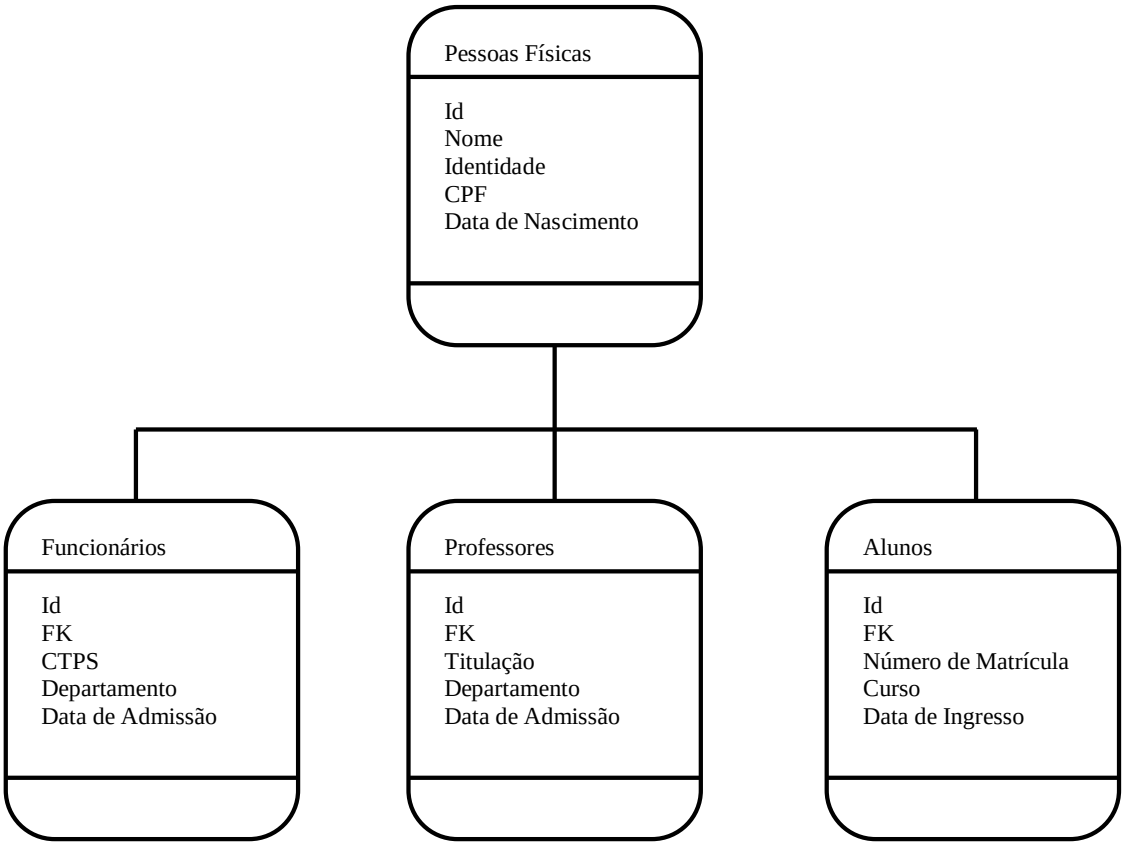


Os *atributos* especificam os requisitos de armazenamento para todas as instâncias do objeto (por exemplo, identificação, fabricante, modelo, ano, capacidade de um objeto, veículo ou nome, e identificação do setor para gerenciador de navegação). As relações entre os objetos são especificadas com o que se chama de ***conexões de instância***, enumeradas com um único número inteiro ou um intervalo, tal como 1 para n, começando em um até o máximo de n. Uma conexão de instância especifica a cardinalidade da relação que um objeto tem com outro. Por exemplo, em uma empresa, o Departamento de Vendas possui ***n*** empregados lotados em suas dependências, mas cada um desses empregados, vendedores, está lotado apenas no Departamento de Vendas; não tem nada a ver, por exemplo, com o Departamento de Produção ou o Departamento de Limpeza e Conservação. Da mesma maneira, nenhum dos empregados lotados nos outros departamentos tem vinculação com departamentos diferentes dos seus.

1.4.2 – Especificação genérica

No caso da **estrutura de especificação genérica**, a classe de objetos é identificada de forma que os objetos que representam instâncias dessa classe herdem os atributos e os serviços fornecidos pelos membros da classe. Por exemplo, em uma

instituição de ensino, o diagrama de especificação de pessoas físicas poderia ficar assim:



O diagrama anterior especifica que funcionários, professores e alunos herdam atributos da classe pessoas físicas, como nome, identidade, CPF, data de nascimento, nome do pai, nome da mãe, etc. Para capturar uma organização de objetos em que alguns deles são componentes (partes) de outro objeto, são introduzidas estruturas todo-parte. Assim, uma pessoa pode trabalhar em uma instituição de ensino, por exemplo, como médico durante o dia e ser professor à noite. Ao mesmo tempo, pode, ainda, ser aluno de um curso de especialização, mestrado ou doutorado oferecido pela mesma instituição. Mas cada um desses cadastros (funcionário, professor, aluno) diz respeito a apenas uma pessoa física.

1.4.3 – Análise orientada a funções

Abordagem funcional para especificação comportamental é mais comumente representada pela combinação de diagrama de fluxo de dados, construções de dados, tabelas de decisão e dicionário de dados, que fazem parte do padrão IEEE para especificação de uma hierarquia funcional.

1.4.4 – Diagrama de Fluxo de Dados

Uma combinação do que são conhecidos por diagrama de fluxo de dados, dicionário de dados e descrição do processo costuma ser utilizada nesta forma de análise de problemas. Um diagrama que especifica os processos (também conhecidos como bolhas, transformações, transações, atividades, operações) e o fluxo de dados entre eles é chamado de Diagrama de Fluxo de Dados (DFD).

Um DFD demonstra formas possíveis de fluxo de informações em um sistema, locais de armazenamento para dados e transformações de dados à medida que eles fluem em um sistema. Os DFDs fornecem uma das mais antigas tecnologias de análise de problemas. São de fácil utilização e muito úteis para preencher a lacuna entre as descrições informais do sistema em um relatório de necessidades e o desenvolvimento das descrições do fluxo de informações no sistema.

As bolhas simbolizam as transformações. Os nomes devem ser exclusivos. Uma seta especifica a direção do fluxo de dados. Os identificadores das setas determinam o tipo dos dados. As setas representam o caminho do fluxo de dados, mas não especificam a ordem dos eventos. Os retângulos (também chamados terminadores) indicam as origens e os destinos dos dados. Finalmente, as linhas paralelas indicam arquivos, bancos de dados, armazenamentos de dados permanentes.

A ideia no uso do DFD é descobrir os fluxos de dados necessários entre as atividades associadas aos objetos em uma partição. O nível mais alto e mais abstrato de um DFD consiste em uma bolha e é chamado de ***nível contexto***.

Inicialmente, um DFD costuma ser de alto nível (omitindo tudo, a não ser os detalhes essenciais), consistindo em uma única bolha. O DFD nível de contexto é então decomposto em um DFD filho. Essa técnica é chamada de nivelamento. Durante o processo de decomposição, deve-se manter a consistência entre os níveis. Isso significa que todo o fluxo da rede de entrada ou de saída de um processo de nível mais alto deve corresponder aos fluxos de entrada e aos fluxos de saída da rede de processos de níveis mais baixos.

1.4.5 – Dicionários de dados

Um dicionário de dados armazena informações sobre os dados encontrados em um DFD. Ele fornece informações, como tipo de dados, acurácia necessária de dados úteis aos projetistas e implementadores:

- Nome, pseudônimo, tipo e descrição indicam como identificar outros nomes possíveis, tipos de dados e o que e como os dados são utilizados, respectivamente.

- Duração, acurácia e intervalo de valores especificam o tempo de vida, a precisão necessária em medidas e todos os valores de dados possíveis, respectivamente.

- Fluxos de dados especificam processos que geram ou recebem dados.

Isso fornece outra ferramenta útil na validação do projeto (caso os requisitos de dados sejam satisfeitos pelo projeto ou pelo código).

No caso de os dados serem provenientes de sistema de tempo real, eles podem ter restrições de tempo que especificam o intervalo de tempo decorrido antes de os dados se tornarem desatualizados. Dessa forma, um dicionário de dados pode ser utilizado para verificar a fidedignidade e a consistência de DFDs. Assim que todas as bolhas, setas e bancos de dados (repositórios) tiverem identificadores e todas as setas tiverem origem e destino, um DFD é considerado completo.

2. Processos de engenharia de requisitos

A Engenharia de Requisitos é um processo que contempla todas as atividades exigidas para criar e manter o documento de requisitos de sistema. Neste contexto, existem quatro atividades genéricas que são consideradas de alto nível: o **estudo da viabilidade**, a **obtenção e análise de requisitos**, a **especificação de requisitos e sua documentação** e, finalmente, a **avaliação desses requisitos**. Porém, praticamente em todos os sistemas se modificam, pois ao longo do tempo as pessoas interessadas desenvolvem melhor compreensão do que elas querem que o software faça; a empresa compradora do sistema sofre modificações; ou, são feitas modificações no hardware, no software e no ambiente organizacional do sistema. Assim, o **gerenciamento de requisitos** é uma atividade adicional da engenharia de requisitos, que se dedica a **gerenciar modificações nos requisitos**.

A engenharia de requisitos pode ser empregada como processo de aplicação de método estruturado, como a análise orientada a objetos. Isso envolve analisar o sistema e desenvolver um conjunto de modelos gráficos de sistema, que, então, atua como especificação de sistema. O conjunto de modelos descreve o comportamento do sistema

e recebe informações adicionais, descrevendo, por exemplo, seu desempenho ou sua confiabilidade.

2.1 – Especificação do processo

A descrição de um processo pode ser escrita em linguagem natural e ter a forma de pseudocódigo "confortável" que possa ser compreendida pelos projetistas. Em um DFD, entradas, dados necessários e ação(ões) são expressos como um único procedimento representado por um círculo com conexões de entrada e saída. Outra abordagem à descrição do processo é o uso de texto do processo (forma de pseudocódigo com palavras-chave no estilo Pascal e estruturas de controle) para descrever o efeito da ação no estado de um modelo ou em possíveis saídas do sistema.

Como a descrição do processo não é feita com a intenção de ser programa de computador (nem mesmo representar um), palavras reservadas de uma linguagem emprestada são utilizadas de maneira informal para fornecer a compreensão e a descrição de como os dados de entrada são transformados em ações específicas. O objetivo da descrição do processo é descrever os procedimentos (processos representados por DFDs). Por exemplo:

Descrição de um processo de exame da LGPD

```
Entrada: m     //Medida com base nos artigos, prazos e punições previstos na
                //LGPD.
        Repetir
                Estimar periodicamente mudanças ocorridas na legislação
                Estimar periodicamente o alcance das normas da LGPD
                Salvar alterações, estimar adequações de Compliance
        Para Repetir
End operação de exame da LGPD
```

Embora os métodos estruturados tenham um papel a desempenhar no processo de engenharia de requisitos, existem muito mais aspectos que os abordados por esses métodos. Dessa forma, são apresentadas a seguir abordagens mais gerais da engenharia de requisitos:

- Estudos de viabilidade

- Levantamento e análise de requisitos

- Validação de requisitos

- Gerenciamento de requisitos

2.1.1 – Abordagem de métodos formais para especificação

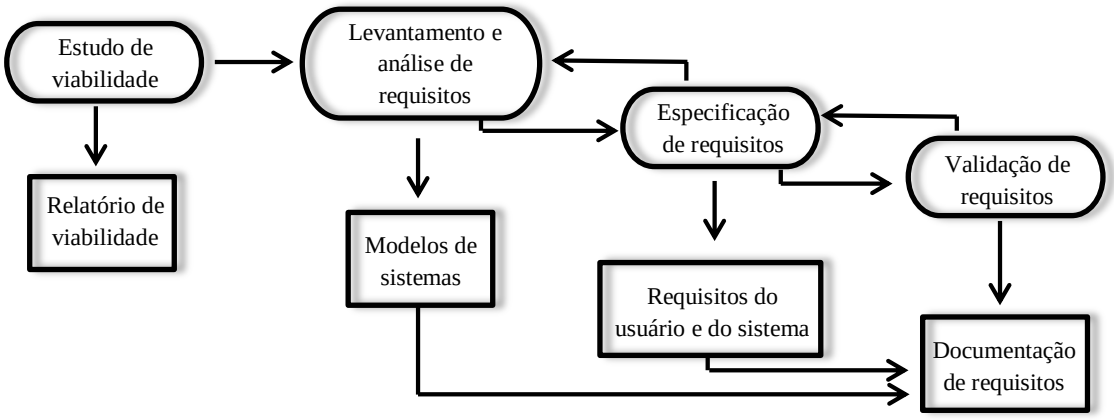
A abordagem dos métodos formais para especificação dos requisitos de software é caracterizada por descrições matemáticas de comportamento. Um ***método formal*** consiste no conjunto de técnicas e notações possíveis de serem expressas matematicamente e que podem ser tratadas de modo mecânico. A motivação para o uso de métodos formais é a obtenção de bases confiáveis para o desenvolvimento de software, caracterizadas pelo raciocínio resultante da aplicação de regras matemáticas.

Um benefício importante da abordagem de métodos formais para a especificação de software é facilitar escrever provas das condições a que o software deve satisfazer. Essa abordagem funciona no desenvolvimento de sistemas críticos de segurança nos quais as falhas podem resultar em prejuízos físicos, de propriedade ou em catástrofes ambientais.

Aliada ao uso intensivo e em larga escala dos métodos formais de desenvolvimento de softwares está a necessidade de atender aos critérios de certificação de qualidade ISO 9000 para produtos de software.

2.2 – Estudos de viabilidade

Para todos os sistemas novos, o processo de engenharia de requisitos de sistema deve começar com o **estudo de viabilidade**, porque se o projeto se mostrar inviável, nem adianta começar. Normalmente, a primeira viabilidade que surge é a econômica. Mas também pode ser relacionada à tecnologia, pessoal qualificado, aspectos legais, dentre outras. A entrada para o estudo de viabilidade é uma descrição geral do sistema e de como ele será utilizado na organização. Os resultados do estudo de viabilidade devem compor um relatório que recomenda se vale a pena ou não realizar o processo de engenharia de requisitos e o processo de desenvolvimento de sistemas. Esquematicamente, fica assim:



Estudo de viabilidade é um levantamento, direcionado, que se destina a responder a algumas perguntas:

1 – O sistema contribui para os objetivos da organização?

2 – O sistema pode ser implementado com a utilização da tecnologia atual dentro das restrições de custo e de prazo?

3 – O sistema pode ser integrado a outros sistemas que já se encontram em operação?

A questão sobre se o sistema contribui ou não para os objetivos da empresa é fundamental. Se não for compatível com esses objetivos, ele não terá nenhum valor real para a organização. Embora isso possa parecer óbvio, muitas organizações desenvolvem sistemas que não contribuem para seus objetivos, seja porque não existe uma declaração clara desses objetivos, seja porque outros fatores políticos ou organizacionais influenciam na aquisição do sistema.

Preparar **estudos de viabilidade** envolve **avaliar e coletar informações** e **redigir relatórios**. A fase de avaliação identifica as informações exigidas para responder às três perguntas apresentadas anteriormente. Uma vez identificadas as informações, é preciso questionar as fontes de informação, a fim de encontrar respostas para essas perguntas. Assim, por exemplo, em relação aos interesses da organização, pode ser questionado:

1 – Como a organização se comportaria se esse sistema fosse implementado?

2 – Quais são os problemas com os processos atuais e como um novo sistema ajudaria a diminuir esses problemas?

3 – Que contribuição direta o sistema trará aos objetivos da empresa?

4 – As informações podem ser transferidas para outros sistemas organizacionais e também podem ser recebidas a partir deles?

5 – O sistema requer tecnologia que não tenha sido utilizada anteriormente na organização?

6 – O que precisa e o que não precisa ser compatível com o sistema?

Entre as fontes de informação que podem responder a essas perguntas, estão os gerentes de departamentos em que o sistema será utilizado, os engenheiros de software que estão familiarizados com o tipo de sistema proposto, peritos em tecnologia, usuários finais de sistemas, etc. Eles devem ser entrevistados durante o estudo de viabilidade, a fim de coletar as informações exigidas.

Quando as informações estiverem disponíveis, o **relatório do estudo de viabilidade** será preparado. Esse relatório deve recomendar se o desenvolvimento do sistema precisa continuar ou não. Ele pode propor mudanças no enfoque, no orçamento e no cronograma, além de sugerir outros requisitos de alto nível para o sistema.

2.3 – Levantamento e análise de requisitos

Depois dos estudos iniciais de viabilidade, o próximo estágio do processo de engenharia de requisitos é o **levantamento e a análise de requisitos**, quando os membros da equipe técnica de desenvolvimento de software trabalham com o cliente e os usuários finais do sistema para descobrir mais informações sobre o domínio da aplicação, que serviços o sistema deve oferecer, o desempenho exigido do sistema, as restrições de hardware e, assim por diante.

O levantamento e a análise de requisitos podem envolver diferentes tipos de pessoas na organização. O termo *stakeholder* é utilizado para se referir a qualquer pessoa que terá ou sofrerá influência direta ou indireta em relação aos requisitos do sistema. Dentre esses *stakeholders* destacam-se os usuários finais que interagirão com o sistema e todo o pessoal, na organização, que venha a ser por ele afetado. Os engenheiros que estão desenvolvendo o sistema ou fazendo a manutenção de sistemas relacionados, os gerentes de negócios, os especialistas nesse domínio, os representantes de sindicato, entre outros, podem também ser os *stakeholders* do sistema.

Levantamento e análise compõem um processo difícil, por diversas razões:

1 – Os *stakeholders* frequentemente não sabem na realidade o que querem do sistema computacional, a não ser em termos muito gerais; eles podem achar difícil articular o que desejam do sistema; eles podem fazer pedidos não realistas, por não terem noção do custo de suas solicitações.

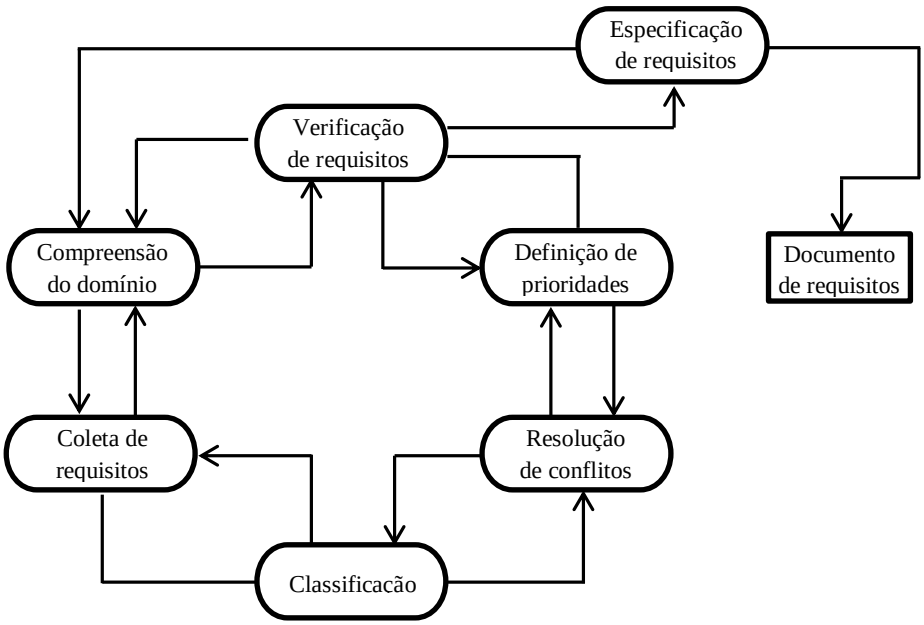
2 – Os *stakeholders* em um sistema expressam naturalmente os requisitos em seus próprios termos e com o conhecimento implícito de sua área de atuação. Os engenheiros de requisitos que não têm experiência no domínio do cliente devem compreender esses requisitos.

3 – Diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneiras distintas. Os engenheiros de requisitos precisam descobrir todas as possíveis fontes de requisitos e encontrar os pontos comuns e os conflitos.

4 – Fatores políticos podem influenciar os requisitos do sistema. Eles podem provir de gerentes que definem requisitos específicos de sistema para aumentar sua influência na organização.

5 – O ambiente econômico e de negócios, no qual a análise de requisitos ocorre, é dinâmico. Inevitavelmente, ele se modifica durante o processo de análise. Como consequência, a importância de requisitos específicos pode mudar. Além disso, novos requisitos podem surgir por parte dos novos *stakeholders*, que não haviam sido consultados inicialmente.

Um modelo genérico do processo de levantamento e análise pode ser esquematicamente montado da seguinte maneira:



Em suma, o levantamento relaciona-se à obtenção dos requisitos do software. Para isso, analistas e engenheiros de software trabalham com clientes e usuários finais para descobrir o problema a ser resolvido, os serviços do sistema, o desempenho necessário, restrições de hardware e outras informações.

Existem técnicas que apoiam as atividades de levantamento de requisitos. Uma breve descrição de algumas delas é:

Entrevista: esta técnica resume-se em "conversas" realizadas com o usuário (entrevistado) para levantar os requisitos do sistema a ser desenvolvido. Podemos decompor esta técnica nas seguintes atividades:

- Ler material de suporte;
- Estabelecer os objetivos da entrevista;
- Decidir quem entrevistar;
- Preparar o entrevistado;
- Decidir os tipos de questões e a sua estrutura.

Uma entrevista pode ser estruturada de três diferentes formas:

Estrutura em pirâmide: iniciam-se as entrevistas com perguntas mais especificas sobre o sistema e se fecham com perguntas mais genéricas. Geralmente utilizadas com usuários mais relutantes;

Estrutura em funil: iniciam-se as entrevistas com perguntas mais genéricas sobre o sistema e se fecham com perguntas mais especificas. Geralmente utilizadas com usuários que tem uma relação mais afetiva com o assunto;

Estrutura em diamante: esta estrutura combina as duas estruturas anteriores e é utilizadas para manter o usuário entrevistado interessado no assunto e para isso se utiliza de perguntas variadas.

Prototipação: é uma versão inicial de um sistema para experimentação. Permite aos utilizadores identificar os pontos fortes e fracos do sistema por ser algo concreto que pode ser criticado. Tem-se dois tipos de protótipos:

Protótipos "Throw-away", que ajudam o levantamento e o desenvolvimento dos requisitos e suportam os requisitos mais difíceis de perceber; e,

Protótipos Evolutivos, que ajudam o desenvolvimento rápido de uma versão inicial do sistema e suportam os requisitos bem definidos e conhecidos.

Algumas das desvantagens da prototipação são os custos de aprendizagem e os custos de desenvolvimento.

JAD (Joint Application Development) é uma técnica que permite a interação entre pessoas que necessitam tomar decisões que afetem múltiplas áreas da organização. Esta técnica envolve atividades de preparação para reuniões, sessões de workshop com os participantes, agenda para as reuniões, participantes assumindo papeis de facilitador / condutor e documentador, além de facilidades visuais, como a utilização de flipchart, quadro negro.

Esta técnica deve ser utilizada nos casos onde existe a necessidade de consenso entre diversos usuários, pois possibilita a todos os envolvidos ter uma visão global do sistema, ajudando a consolidar interesses de diversos usuários quanto ao sistema a ser desenvolvido.

O objetivo dessa técnica é aumentar o comprometimento e participação do usuário e obter subsídios para elaborar o documento de Especificação de Requisitos para o sistema com consenso de todos, de forma a ser uma validação formal dos requisitos do sistema.

Cada organização pode ter sua própria versão ou uma versão mais definida do esquema acima proposto, dependendo de fatores locais, como perícia da equipe, tipo de sistema em desenvolvimento, padrões utilizados, entre outros. As atividades de processo são:

1 – ***Compreensão do domínio***: os analistas devem desenvolver sua compreensão do domínio da aplicação. Por exemplo, se for exigido um sistema para supermercado, o analista deverá ser capaz de descobrir como operam os supermercados.

2 – ***Coleta de requisitos***: é preciso interagir com os *stakeholders* do sistema para descobrir seus requisitos. Obviamente, a compreensão do domínio se desenvolve mais durante essa atividade.

3 – ***Classificação***: essa atividade considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes.

4 – ***Resolução de conflitos***: essa atividade se ocupa em encontrar e solucionar conflitos, pois quando múltiplos *stakeholders* estão envolvidos, os requisitos naturalmente apresentam conflitos de interesses.

5 – ***Definição das prioridades***: em qualquer conjunto de requisitos, alguns serão mais importantes que outros. Esse estágio envolve a interação com os *stakeholders*, para descobrir os requisitos mais importantes.

6 – ***Verificação de requisitos***: os requisitos são verificados a fim de se descobrir se eles são completos e consistentes e se estão em concordância como que os *stakeholders* realmente desejam do sistema.

Assim, levantamento e análise dos requisitos é um processo iterativo, com feedback contínuo de cada atividade para as outras. **O ciclo começa com a compreensão do domínio e termina com a verificação dos requisitos**. A compreensão dos requisitos pelo analista melhora a cada fase do ciclo.

Não existe uma **abordagem perfeita e universalmente aplicável** para a análise de requisitos. Normalmente, **é preciso utilizar várias abordagens** para desenvolver compreensão e análise completa dos requisitos, mas três técnicas muito importantes no desenvolvimento desse trabalho são: o **levantamento orientado a pontos de vista**; os

cenários e a **etnografia**. Outras técnicas que podem ser utilizadas são os métodos de análise estruturada e a prototipação.

2.3.1 – Levantamento orientado a pontos de vista

Para qualquer sistema, de tamanho médio ou grande, normalmente há diferentes tipos de usuário final, o que implica em que muitos *stakeholders* têm algum tipo de interesse nos requisitos de sistema. Por exemplo, considere um terminal ATM (Automatic Teller Machine), o famoso caixa eletrônico de banco, e todos os *stakeholders* de sistema envolvidos. Uma análise preliminar aponta para, pelo menos:

1 – Clientes do banco que recebem os serviços do sistema;

2 – Representantes de outros bancos que têm acordos de reciprocidade que permitam o uso de ATMs uns dos outros;

3 – Gerentes de agências bancárias que obtêm informações do sistema;

4 – Equipes de atendimento de balcão em agências bancárias envolvidas na operação diária do sistema, que atendem às demandas dos clientes;

5 – Administradores de bancos de dados que são responsáveis pela integração do sistema com o banco de dados do cliente do banco;

6 – Gerentes de segurança bancária que devem assegurar que o sistema não apresente nenhum tipo de falha de segurança;

7 – Departamento de marketing do banco que provavelmente estará interessado em utilizar o sistema como um instrumento de marketing do banco; e,

8 – Engenheiros de manutenção de hardware e software que têm responsabilidade de fazer a manutenção e a atualização de hardware e software.

Essa relação mostra que, mesmo para um sistema relativamente simples, existem **muitos pontos de vista diferentes** que devem ser considerados. Diferentes ***pontos de vista*** a respeito de um problema “veem” o problema de modos diferentes. Contudo, suas perspectivas não são inteiramente independentes, mas em geral apresentam alguma duplicidade, de modo que se conectam através de requisitos comuns.

Abordagens orientadas a pontos de vista, na Engenharia de Requisitos, reconhecem esses diferentes pontos de vista e os utilizam para estruturar e organizar o processo de levantamento e os próprios requisitos. Uma importante capacidade da análise orientada a pontos de vista é que ela reconhece a existência de várias perspectivas e oferece *framework* para descobrir conflitos nos requisitos propostos por diferentes *stakeholders*.

Por outro lado, diferentes métodos apresentam diferentes ideias sobre o que significa “ponto de vista”. Isso pode ser feito considerando:

1 – ***Uma fonte ou “drenos” de dados***: nesse caso, os pontos de vista são responsáveis pela produção ou consumo de dados. A análise envolve identificar todos esses pontos de vista, identificar quais dados são produzidos ou consumidos e que processamento é realizado.

2 – ***Um framework de representação***: nesse caso, um ponto de vista é considerado um tipo particular de modelo de sistema. Por exemplo, diferentes engenheiros devem desenvolver um modelo de relacionamento de entidades, um modelo de máquina de estados, entre outros. Cada abordagem de análise descobre diferentes aspectos sobre o sistema que está sendo analisado.

3 – ***Um receptor de serviços***: nesse caso, os pontos de vista são externos ao sistema e dele recebem serviços. Eles podem fornecer dados para esses serviços ou sinais de controle. A análise envolve examinar os serviços recebidos por diferentes pontos de vista, coletando esses serviços e resolvendo conflitos.

Cada um desses modelos de ponto de vista apresenta diferentes aspectos positivos e negativos. Os pontos de vista como fontes de dados ou “drenos” e os pontos de vista como representações são particularmente valiosos para a descoberta de conflitos detalhados entre os requisitos. Contudo, eles são menos adequados para estruturar o processo de análise de requisitos, uma vez que não há nenhuma relação simples entre os pontos de vista e os *stakeholders* do sistema.

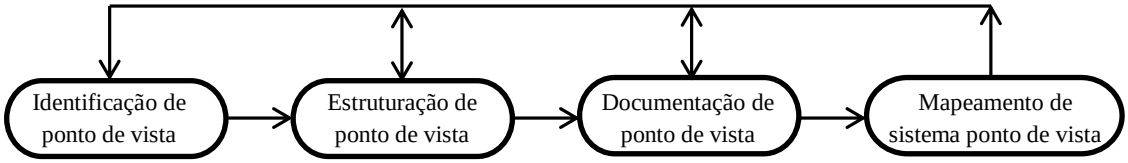
Sistemas interativos fornecem serviços a usuários finais. Consequentemente, a abordagem mais eficaz orientada a pontos de vista, para a análise de sistemas interativos, utiliza pontos de vista externos. Esses pontos de vista interagem com o sistema, recebendo serviços dele e fornecendo dados e sinais de controle para ele. As vantagens desse tipo de ponto de vista são:

1 – Os pontos de vista são externos ao sistema e, assim, são uma maneira natural de estruturar o processo de levantamento de requisitos.

2 – É relativamente fácil decidir se alguma coisa é um ponto de vista válido. Os pontos de vista devem interagir com o sistema de alguma maneira.

3 – Os pontos de vista e os serviços são um meio útil de estruturar os requisitos não funcionais. Cada serviço pode ter requisitos não funcionais associados. Os pontos de vista múltiplos permitem que o mesmo serviço tenha diferentes requisitos não funcionais em diferentes pontos de vista.

Por sua vez, o método VORD (*Viewpoint-Oriented Requirements Definition* – Definição de Requisitos Orientada a Pontos de Vista) foi projetado como um framework orientado a serviços, para levantamento e análise de requisitos. Esquematicamente, ele apresenta a seguinte disposição:



Os principais estágios do método VORD são:

1 – **Identificação de pontos de vista**, que envolve descobrir os pontos de vista que utilizam serviços do sistema e identificar os serviços específicos fornecidos para cada ponto de vista.

2 – **Estruturação de pontos de vista**, que envolve agrupar pontos de vista relacionados, segundo uma hierarquia. Serviços comuns são localizados nos níveis mais altos da hierarquia e herdados por pontos de vista de nível inferior.

3 – **A documentação do ponto de vista**, que envolve refinar a descrição dos pontos de vista e serviços identificados.

4 – **Mapeamento de sistema conforme pontos de vista**, que envolve identificar objetos em um projeto orientado a objetos, utilizando as informações de serviço que estão encapsulados nos pontos de vista.

As informações sobre o ponto de vista no VORD são coletadas utilizando-se formulários-padrão. O VORD também utiliza várias notações diagramáticas, inclusive diagramas de hierarquia de ponto de vista e cenários de eventos.

A primeira etapa da análise de ponto de vista é identificar os possíveis pontos de vista. Como em todos os métodos, essa identificação inicial é provavelmente o estágio mais difícil. Uma das abordagens é o *brainstorming*, no qual se identificam os serviços em potencial e as entidades que interagem com o sistema. Os *stakeholders* se reúnem e sugerem possíveis pontos de vista, que são anotados em um diagrama de bolhas, no qual os pontos de vista em potencial são mostrados em bolhas separadas.

Em uma reunião de *brainstorming*, devem-se tentar identificar os pontos de vista em potencial, os serviços do sistema, as entradas de dados, os requisitos não funcionais, os eventos de controle e exceções. Nesse estágio da análise, não se deve tentar impor uma estrutura ao diagrama. As fontes de informações que podem ser utilizadas para criar essa visão inicial do sistema podem ser os documentos que estabelecem os

objetivos de alto nível do sistema, o conhecimento de engenheiros de software sobre projetos precedentes ou a experiência como clientes. As entrevistas podem ser feitas com gestores, encarregados, consultores, engenheiros e clientes.

Além de receber serviços, os pontos de vista também fornecem entradas para esses serviços. Por outro lado, os pontos de vista também fornecem informações sobre controle, a fim de determinar se e quando os serviços serão efetuados. Durante esse estágio inicial do processo, essas informações de dados e controle são simplesmente identificadas pelo nome.

As informações de ponto de vista são utilizadas para preencher os *templates* de ponto de vista e organizar os pontos de vista em uma hierarquia de herança. Para mostrar pontos de vista em comum e reutilizar as informações do ponto de vista, os fatores de hierarquia de herança salientam os pontos de vista que fornecem serviços comuns. As informações de serviços, dados e controle são herdadas por subpontos de vista.

O próximo estágio do processo é descobrir mais informações detalhadas sobre os serviços oferecidos, os dados que eles requerem e como são controlados. Os requisitos são coletados em consulta aos *stakeholders* associados com cada ponto de vista. Serão discutidas as necessidades de serviço de cada ponto de vista diferente, seja com os usuários finais ou com os especialistas em pontos de vista, se o ponto de vista for outro sistema automatizado.

***Templates* de pontos de vista e de serviços**, assim como **cenários de eventos**, são desenvolvidos para todos os pontos de vista e serviços identificados. As informações podem, então, sofrer verificação cruzada para descobrir erros na análise e conflitos de requisitos. Como isso gera grande quantidade de informações, o VORD, como outros métodos de análise, somente é de uso prático com o apoio de ferramentas CASE.

2.3.2 – Cenários

As pessoas geralmente acham mais fácil relacionar exemplos da vida real que descrições abstratas. Elas podem compreender e criticar um cenário de como poderiam interagir com um sistema de software. Os engenheiros de requisitos podem utilizar as informações obtidas com essa discussão para formular os requisitos reais do sistema.

Assim, os cenários podem ser particularmente úteis para acrescentar detalhes a um esboço da descrição de requisitos. Estas são descrições de exemplos de sessões de interação. Cada cenário aborda uma interação ou um pequeno número de possíveis interações. Diferentes tipos de cenários podem ser desenvolvidos, de modo a fornecerem diferentes tipos de informações, com diferentes níveis de detalhes sobre o sistema.

O cenário começa com um esboço de interação e, durante o levantamento de requisitos, são acrescentados detalhes para criar uma descrição completa dessa interação. De forma geral, o cenário pode incluir:

1 – Descrição do estado da arte do sistema no início do cenário;

2 – Descrição do fluxo normal de eventos no cenário;

3 – Descrição do que pode sair errado e de como lidar com isso;

4 – Informações sobre outras atividades que possam estar em andamento ao mesmo tempo; e,

5 – Descrição do estado da arte do sistema no final do cenário.

A obtenção de requisitos com base em cenários pode ser realizada informalmente, e o engenheiro de requisitos trabalha com os *stakeholders* para identificar cenários e captar detalhes desses cenários. Como alternativa, pode ser empregada uma abordagem mais estruturada, como os cenários de eventos ou use-cases.

2.3.2.1 – Cenários de eventos

Os cenários são utilizados em VORD para documentar o comportamento do sistema quando apresentado com eventos específicos. Cada evento distinto de interação, como inserir, por exemplo, um cartão de banco no terminal ATM ou selecionar um serviço ATM, pode ser documentado com um cenário de evento separado. Entre os cenários de eventos estão a descrição do fluxo de dados e as ações do sistema. Além disso, esses cenários documentam as exceções que possam surgir.

As convenções diagramáticas utilizadas em cenários de eventos são:

1 – Os dados fornecidos a partir de um ponto de vista ou entregues a um ponto de vista são mostrados em elipses;

2 – As informações de controle entram e saem da parte superior de cada bloco;

3 – Os dados saem da direita de cada bloco. Se eles não forem mencionados, isso significará que são internos ao sistema;

4 – As exceções são mostradas na parte inferior do bloco. Onde houver diversas opções possíveis, elas serão incluídas em um bloco; e,

5 – O nome do próximo evento esperado, depois do término do cenário, é mostrado em um bloco sombreado.

2.3.2.2 – Casos de uso

Os casos de uso são técnicas baseadas em cenários para a obtenção de requisitos, que foram introduzidos pela primeira vez no método Obrigatory. Eles se tornaram uma

característica fundamental da notação em UML (Unified Modeling Language – Linguagem de Modelagem Unificada), para descrever modelos de sistemas orientados a objetos. Em sua forma mais simples, um caso de uso identifica os agentes envolvidos numa interação e especifica o tipo de interação.

Os agentes no processo são representados por bonecos e cada classe de interação é representada por uma elipse com um nome. O conjunto de casos de uso representa todas as possíveis interações que serão representadas nos requisitos de sistema.

Às vezes, existe confusão sobre se um caso de uso é ou não um cenário em si; ou como foi sugerido por Fowler, um caso de uso engloba um conjunto de cenários em que cada cenário é um traço isolado dentro de um caso de uso. Nesse caso, haveria um cenário para a interação normal, mais os cenários para cada exceção possível.

Dentro do UML, diagramas de sequências podem ser utilizados para acrescentar informações a um caso de uso. Esses diagramas de sequência mostram os agentes envolvidos na interação, os objetos dentro do sistema com os quais eles interagem e as operações que estão associadas a esses objetos.

A UML é um verdadeiro padrão para a modelagem orientada a objetos; assim, os casos de uso e a obtenção de requisitos com base em casos de uso são cada vez mais utilizados para obter requisitos.

2.3.3 – Etnografia

Os sistemas de software não existem isoladamente. Eles são utilizados em um contexto social e organizacional. Os requisitos de sistema de software podem ser derivados ou limitados por esse contexto. Frequentemente, satisfazer a esses requisitos sociais e organizacionais é fundamental para o sucesso do sistema. **Uma razão pela qual muitos sistemas de software são entregues, mas nunca utilizados, é que eles não consideram a importância desse tipo de requisito de sistema**.

A etnografia é uma técnica de observação que pode ser utilizada para compreender os requisitos sociais e organizacionais. Um analista se insere no ambiente de trabalho em que o sistema será utilizado e o trabalho diário é observado, momento em que são anotadas as tarefas reais em que os participantes então envolvidos. O valor da etnografia é que ela ajuda a descobrir requisitos de sistema implícitos, que refletem os processos reais, em vez dos processos formais, em que as pessoas estão envolvidas.

Os usuários geralmente acham muito difícil articular detalhes de seu trabalho, porque o trabalho para eles é sua segunda natureza. Eles compreendem seu próprio trabalho, mas podem não compreender a relação dele com outras atividades na organização. Os fatores sociais e organizacionais que afetam o trabalho, mas não são óbvios para os indivíduos, somente podem ficar claros quando examinados por um observador imparcial.

Geralmente, as práticas de trabalho reais são muito mais ricas, complexas e dinâmicas que os simples modelos supostos pelos sistemas de automação de escritórios. A diferença entre o trabalho suposto e o real, provavelmente é a razão mais importante pela qual, sistemas de escritório não têm nenhum efeito significativo na produtividade. A etnografia é particularmente eficaz na descoberta de dois tipos de requisitos:

1 – Os requisitos derivados da maneira como as pessoas realmente trabalham, em vez da maneira pela qual as definições de processo dizem como elas deveriam trabalhar; e,

2 – Os requisitos derivados da cooperação e conscientização das atividades de outras pessoas.

A etnografia pode ser combinada com a prototipação. A etnografia instrui o desenvolvimento do protótipo, de modo que um número menor de ciclos de treinamento do protótipo seja necessário. Além disso, a prototipação enfoca a etnografia pela identificação de problemas e questões que podem então ser discutidas com o etnógrafo, o qual deve, dessa forma, procurar respostas para essas questões durante a nova fase de estudo do sistema.

Os estudos de etnografia podem revelar importantes detalhes de processo que, frequentemente, são omitidos por outras técnicas de obtenção de requisitos. Contudo, devido ao seu enfoque no usuário final, essa abordagem não é apropriada para descobrir requisitos organizacionais e de domínio. Ela nem sempre pode identificar novas características que deveriam ser acrescentadas ao sistema. Neste sentido, a etnografia não é, portanto, uma abordagem completa para a obtenção de requisitos e deveria ser utilizada com outras abordagens, como a análise de casos de uso.

2.4 – Validação de requisitos

A validação de requisitos se ocupa de mostrar que os requisitos realmente definem o sistema que o cliente deseja. Ele tem muito em comum com a análise de requisitos, uma vez que se preocupa em descobrir problemas nos requisitos. Contudo, esses são processos distintos, já que a validação deve se ocupar da elaboração de um esboço completo do documento de requisitos, enquanto a análise envolve trabalhar com requisitos incompletos.

A validação de requisitos é importante porque a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao retrabalho, quando esses erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação. O custo de fazer modificações no sistema, resultantes de problemas de requisitos, é muito maior que reparar erros de projeto ou de codificação. A razão disso é que mudanças nos requisitos, em geral, significam que o projeto do sistema e a

implementação também devem ser modificados e que o sistema tem de ser novamente testado.

Durante o processo de validação de requisitos, diferentes tipos de verificação devem ser realizados sobre os requisitos constantes do documento de requisitos. Dentre essas verificações, destacam-se:

1 – ***Verificações de validade***: o usuário pode pensar que um sistema é necessário para realizar certas funções. Contudo, mais estudos e análises podem identificar funções adicionais ou diferentes, que também são exigidas. Os sistemas têm diversos usuários com necessidades diferentes e qualquer conjunto de requisitos é inevitavelmente uma solução conciliatória da comunidade de usuários.

2 – ***Verificações de consistência***: os requisitos em um documento não devem ser conflitantes, ou seja, não devem existir restrições contraditórias ou descrições diferentes para uma mesma função do sistema.

3 – ***Verificações de completeza***: o documento de requisitos deve incluir requisitos que definam todas as funções e restrições exigidas pelo usuário do sistema.

4 – ***Verificações de realismo***: utilizando o conhecimento da tecnologia existente, os requisitos devem ser verificados, a fim de assegurar que eles realmente podem ser implementados. Essas verificações devem também levar em conta o orçamento e os prazos para o desenvolvimento do sistema.

5 – ***Facilidade de verificação***: para reduzir o potencial de divergências entre clientes e fornecedores, os requisitos do sistema devem ser escritos de modo que possam ser verificados. Isso significa que um conjunto de verificações pode ser projetado para mostrar que o sistema entregue cumpre com esses requisitos.

Existe uma série de técnicas de validação de requisitos que podem ser utilizadas em conjunto ou individualmente:

1 – ***Revisões de requisitos***: os requisitos são analisados sistematicamente por uma equipe de revisores.

2 – ***Prototipação***: nessa abordagem de validação, um modelo executável do sistema é mostrado aos usuários finais e clientes. Eles podem experimentar o modelo para verificar se ele atende às suas necessidades.

3 – ***Geração de casos de teste***: como modelo ideal, os requisitos deveriam ser testáveis. Se os testes para os requisitos são criados como parte do processo de validação, isso, muitas vezes, revela problemas com os requisitos. Se um teste é difícil ou impossível de ser projetado, isso frequentemente significa que os requisitos serão de difícil implementação e devem ser reconsiderados.

4 – ***Análise automatizada da consistência***: se os requisitos são expressos como modelos de sistema em notação estruturada ou formal, então as ferramentas CASE

podem ser utilizadas para verificar a consistência do modelo. Para verificar a consistência, a ferramenta CASE deve construir uma base de dados de requisitos e, então, utilizando-se as regras do método ou da notação, verificar todos os requisitos na base de dados. A análise de requisitos produz um relatório das inconsistências que forem descobertas.

As dificuldades de validação de requisitos não devem ser subestimadas. É difícil demonstrar que um conjunto de requisitos atende às necessidades de um usuário. Os usuários devem pensar no sistema em operação e imaginar como esse sistema se adequaria ao seu trabalho. Não é fácil para profissionais habilitados de computação conseguir realizar esse tipo de análise abstrata, o que é ainda mais difícil para os usuários de sistema. Como resultado, a validação de requisitos raramente descobre todos os problemas com os requisitos, e as modificações para corrigir omissões e falhas de compreensão, depois que o documento de requisitos foi aceito, são inevitáveis.

2.4.1 – Revisões de requisitos

Uma revisão de requisitos é um processo manual, que envolve muitos leitores, tanto do pessoal do cliente como do fornecedor, que verificam o documento de requisitos a fim de detectar anomalias ou omissões. O processo de revisão deve ser gerenciado da mesma maneira que as inspeções de programa. Como alternativa esse processo pode ser organizado em maior escala, com muitos participantes envolvidos na verificação de diferentes partes do documento.

As revisões de requisitos podem ser informais ou formais. As revisões informais simplesmente envolvem os fornecedores que discutem os requisitos com tantos *stakeholders* quantos forem possíveis. É surpreendente como muitas vezes a comunicação entre desenvolvedores e *stakeholders* termine depois da obtenção de requisitos, e não exista nenhuma confirmação de que os requisitos documentados são os que os *stakeholders* realmente solicitaram. Muitos problemas podem ser detectados simplesmente conversando sobre o sistema com os *stakeholders*, antes de iniciar uma revisão formal.

Em uma revisão formal de requisitos, a equipe de desenvolvimento deve "conduzir" o cliente pelos requisitos de sistema, explicando as implicações de cada um. A equipe de revisão deve verificar cada requisito, em termos de sua consistência, e checar os requisitos como um todo, sob o ponto de vista de sua completeza. Os revisores também podem checar:

1 – ***Facilidade de verificação***: o requisito é realmente passível de ser testado, como foi definido?

2 – ***Facilidade de compreensão***: o requisito pode ser adequadamente compreendido pelos compradores ou usuários finais do sistema?

3 – ***Facilidade de rastreamento***: a origem do requisito é claramente definida? (Pode ser preciso retornar à origem do requisito, a fim de avaliar o impacto de uma mudança. A facilidade de rastreamento é importante porque permite avaliar o impacto de uma mudança no restante do sistema).

4 – ***Adaptabilidade***: o requisito é adaptável? (Em outras palavras, ele pode ser modificado, sem que isso provoque efeitos de grande escala em outros requisitos do sistema?).

Conflitos, contradições, erros e omissões nos requisitos devem ser destacados durante a revisão e formalmente registrados. Fica, então, por conta dos usuários, do comprador do sistema e dos desenvolvedores negociar a solução para esses problemas identificados.

2.5 – Gerenciamento de requisitos

Os requisitos para grandes sistemas de software estão sempre sendo modificados. Uma das razões para isso é que esses sistemas são geralmente desenvolvidos para lidar com problemas “intrincados”. Como o problema não pode ser inteiramente definido, os requisitos de sistema são necessariamente incompletos. Durante o processo de desenvolvimento do software, a compreensão dos desenvolvedores sobre o problema está constantemente se modificando, e essas mudanças se refletem nos requisitos.

Grandes sistemas de software são geralmente necessários para melhorar o status atual, pois o sistema existente pode ser desatualizado. Embora as dificuldades com o sistema atual possam ser conhecidas, é difícil prever que efeitos o sistema “aperfeiçoado” terá sobre a organização. Depois que os usuários finais se familiarizam com um sistema, novos requisitos surgem, geralmente pelas seguintes razões:

1 – Grandes sistemas geralmente têm uma comunidade de usuários bastante diversificada. Diferentes usuários têm diferentes prioridades, que podem ser conflitantes ou contraditórias. Os requisitos finais do sistema são, inevitavelmente, uma conciliação entre eles; e, com a experiência, muitas vezes é constatado que o equilíbrio do apoio dado a diferentes usuários precisa ser mudado.

2 – O pessoal que paga pelo sistema e os usuários desse sistema raramente são as mesmas pessoas. Os clientes do sistema impõem requisitos em razão de restrições organizacionais e orçamentárias, e esses requisitos podem ser conflitantes com os requisitos dos usuários finais.

3 – A empresa e o ambiente técnico do sistema se modificam, e isso tem de ser refletido no próprio sistema. Um novo hardware pode ser implementado; pode ser necessário fazer a interface do sistema com outros sistemas; as prioridades da empresa podem se modificar, acarretando consequentes mudanças no suporte necessário ao sistema; e, novas legislações e regulamentos podem ser criados, e estes devem ser implementados pelo sistema. Os requisitos não funcionais são, particularmente, afetados por mudanças na tecnologia de hardware.

Dessa forma, o gerenciamento de requisitos se constitui no processo de **compreender e controlar as mudanças nos requisitos de sistemas**. O processo de gerenciamento de requisitos é realizado em conjunto com outros processos da engenharia de requisitos. O planejamento se inicia ao mesmo tempo em que o levantamento inicial de requisitos, e o gerenciamento ativo dos requisitos deve começar assim que um esboço da versão do documento de requisitos estiver disponível.

2.5.1 – Requisitos permanentes e voláteis

O desenvolvimento de requisitos de software focaliza a atenção nos recursos do hardware, nos objetivos da empresa e em outros sistemas da empresa. Enquanto a definição de requisitos é desenvolvida, uma melhor compreensão das necessidades dos usuários é alcançada. Isso fornece informações para o usuário, causando modificações nos requisitos. Pode levar vários anos para se especificar e desenvolver um sistema de grande porte. Ao longo do tempo, o ambiente do sistema e os objetivos da empresa certamente deverão ser modificados. Os requisitos devem, portanto, evoluir, a fim de refletir essas mudanças.

Partindo de uma perspectiva de evolução, os requisitos podem ser divididos em duas classes:

1 – ***Requisitos permanentes***: são requisitos relativamente estáveis, que derivam da atividade principal da organização e que se relacionam diretamente com o domínio do sistema. Por exemplo, em um hospital, sempre haverá requisitos relativos a pacientes, médicos, enfermeiras e tratamentos. Esses requisitos podem ser derivados dos modelos de domínio, que mostram as entidades e os relacionamentos que caracterizam um domínio de aplicação.

2 – ***Requisitos voláteis***: são requisitos que provavelmente vão se modificar durante o desenvolvimento do sistema ou depois que o sistema estiver em operação, como aqueles resultantes de políticas governamentais sobre assistência médica. Os requisitos voláteis também podem ser classificados em:

Tipo de requisito	Descrição
Requisitos mutáveis	Requisitos que se modificam por causa das mudanças no ambiente no qual a organização está operando. Por exemplo, em sistemas de hospitais, o financiamento do tratamento de pacientes pode se modificar e, assim, exigir que diferentes informações sobre o tratamento sejam coletadas.
Requisitos emergentes	Requisitos que surgem à medida que a compreensão do cliente do sistema evolui durante o desenvolvimento do sistema. O processo de projeto pode revelar novos requisitos emergentes.
Requisitos consequentes	Requisitos que resultam da introdução do sistema de computação. A introdução do sistema de computação pode modificar os processos da organização e criar novos meios de trabalho, que geram novos requisitos de sistema.
Requisitos de compatibilidade	Requisitos que dependem de sistemas ou processos de negócios específicos dentro da organização. À medida que eles se modificam, os requisitos de compatibilidade no sistema encomendado ou entregue podem também ter de evoluir.

2.5.2 – Planejamento do gerenciamento de requisitos

Planejar é o primeiro estágio essencial ao processo de gerenciamento de requisitos, que é muito dispendioso; e para cada projeto, o estágio de planejamento estabelece o nível de detalhes exigido para o gerenciamento de requisitos. Durante o estágio de gerenciamento de requisitos, tem-se que decidir sobre os seguintes aspectos:

1 – ***Identificação de requisitos***: cada requisito precisa ser identificado de modo único, para que possa ser feita referência cruzada deste com os outros requisitos e para que ele possa ser utilizado nas avaliações de facilidade de rastreamento.

2 – ***Processo de gerenciamento de mudanças***: trata-se do conjunto de atividades que avalia o impacto e o custo das mudanças.

3 – ***Políticas de facilidade de rastreamento***: definem as relações entre os requisitos; entre requisitos e o projeto de sistema que devem ser registrados; e, também como esses registros devem ser mantidos.

4 – ***Suporte de ferramentas CASE***: o gerenciamento de requisitos envolve processar grande quantidade de informações sobre requisitos. As ferramentas que podem ser utilizadas, vão desde sistemas especializados em gerenciamento de requisitos até planilhas de cálculo e sistemas simples de bancos de dados.

Existem muitas relações entre requisitos e outros requisitos, e entre os requisitos e o projeto do sistema. Mas há também elos entre os requisitos e as razões básicas da proposição desses requisitos. Quando são propostas modificações, é preciso verificar o impacto dessas mudanças sobre outros requisitos e o projeto de sistema. A facilidade de rastreamento é propriedade geral de uma especificação de requisito que reflete a facilidade de se encontrar requisitos relacionados.

Existem três tipos de informações sobre a facilidade de rastreamento, que podem ser mantidas:

1 – ***Informações sobre a facilidade de rastreamento da origem*** vinculam os requisitos aos *stakeholders* que propuseram esses requisitos. Quando uma mudança é proposta, essa informação é utilizada para descobrir os *stakeholders*, de modo que eles possam ser consultados sobre a mudança.

2 – ***Informações sobre a facilidade de rastreamento de requisitos*** vinculam requisitos dependentes dentro do seu respectivo documento. Essa informação é utilizada para avaliar quantos requisitos provavelmente serão afetados pela mudança proposta e a extensão das mudanças consequentes nos requisitos, que podem ser necessárias.

3 – ***Informações sobre a facilidade de rastreamento de projeto*** vinculam os requisitos aos módulos de projeto em que esses requisitos são implementados. Essa informação é utilizada para avaliar o impacto das mudanças nos requisitos, as propostas para o projeto e a implementação do sistema.

Informações sobre a facilidade de rastreamento são, frequentemente, representadas com o uso de matrizes de facilidade de rastreamento, que relacionam os requisitos aos *stakeholders*, os requisitos entre si ou aos módulos de projeto. Se forem consideradas as matrizes de facilidade de rastreamento que vinculam os requisitos a outros requisitos, cada requisito será representado por uma linha e por uma coluna na matriz. Onde existe dependência entre requisitos, ela é registrada na célula, na interseção linha/coluna.

Assim, a letra U na interseção linha/coluna ilustra que o requisito na linha utiliza os recursos especificados no requisito nomeado na coluna; enquanto R significa que existe uma fraca relação entre os requisitos. Por exemplo, eles podem ambos definir os requisitos para partes do mesmo subsistema.

As matrizes de facilidade de rastreamento podem ser utilizadas quando um pequeno número de requisitos precisa ser gerenciado, mas em grandes sistemas com muitos requisitos, elas se tornam muito difíceis de serem trabalhadas e são de manutenção dispendiosa. Para esses sistemas, tem-se que obter as informações da facilidade de rastreamento num banco de dados de requisitos, em que cada requisito é explicitamente vinculado a requisitos relacionados. O impacto das mudanças pode, então, ser avaliado pelo uso dos recursos de visualização dos bancos de dados. Como alternativa, pode ser possível gerar matrizes de facilidade de rastreamento automaticamente.

O gerenciamento de requisitos precisa de algum apoio automatizado e as ferramentas CASE utilizadas devem ser escolhidas durante a fase de planejamento. O apoio de ferramentas é necessário para:

1 – ***Armazenamento de requisitos***: os requisitos devem ser mantidos em um depósito de dados seguro, gerenciado, que seja acessível a todos os envolvidos no processo de engenharia de requisitos.

2 – ***Gerenciamento de mudanças***: esse processo é simplificado se o apoio de ferramentas ativas estiver disponível.

3 – ***Gerenciamento de facilidade de rastreamento***: o apoio de ferramentas para a rastreabilidade permite que sejam descobertos requisitos relacionados. Existem várias ferramentas que utilizam técnicas de processamento de linguagem natural para ajudar a descobrir as possíveis relações entre os requisitos.

Para sistemas pequenos, pode não ser necessário utilizar ferramentas especializadas de gerenciamento de requisitos. Esse processo pode ter apoio utilizando os recursos disponíveis em processadores de texto, planilhas de cálculo e bancos de dados de PCs. Contudo, para sistemas maiores, é necessário o apoio de ferramentas mais especializadas.

2.5.3 – Gerenciamento de mudanças de requisitos

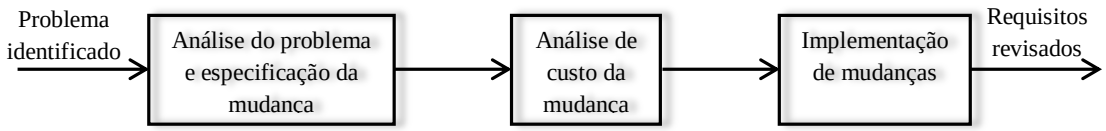
O gerenciamento de mudanças de requisitos deve ser aplicado a todas as mudanças propostas aos requisitos. A vantagem de se utilizar um processo formal para o gerenciamento de mudanças é que todas as propostas de mudanças são tratadas de modo consistente, e as mudanças no documento de requisitos são feitas de maneira controlada. Há três estágios principais em um processo de gerenciamento de mudanças:

1 – ***Análise do problema e especificação da mudança***: o processo começa com a identificação de um problema com os requisitos ou, algumas vezes, com a proposta específica de mudança. Nesse estágio, é realizada a análise do problema ou a proposta de mudança, a fim de verificar sua validade. Uma proposta mais específica de mudança nos requisitos pode então ser efetuada.

2 – ***Análise e custo da mudança***: o efeito da mudança proposta é avaliado, utilizando-se informações sobre a facilidade de rastreamento e o conhecimento geral dos requisitos do sistema. O custo da mudança é estimado em termos das modificações no documento de requisitos e, se apropriado, no projeto de sistemas e na implementação. Uma vez concluída esta análise, é tomada a decisão sobre prosseguir com a alteração de requisitos ou não.

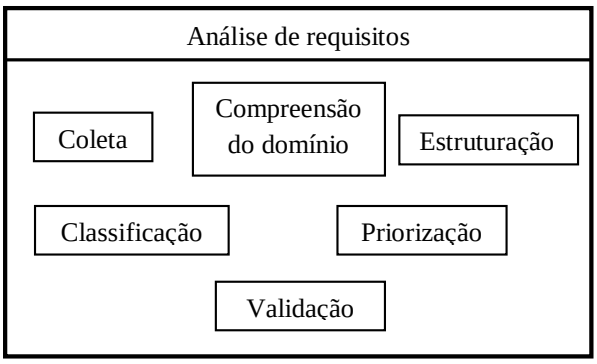
3 – ***Implementação de mudanças***: o documento de requisitos e, quando for necessário, o projeto de sistema e a implementação são modificados. O documento de requisitos deve ser organizado de maneira que as mudanças possam ser acomodadas sem muito esforço. Assim, como ocorre com os programas, a facilidade de modificações em documentos é alcançada ao se minimizar referências externas e ao tornar as seções do documento tão modulares quanto possível.

Esquematicamente, temos o gerenciamento de mudanças de requisitos estruturado da seguinte maneira:



Se uma mudança nos requisitos do sistema for requerida urgentemente, sempre existe a tentação de fazer essa mudança no sistema e, depois, retrospectivamente, modificar o documento de requisitos. Isso quase inevitavelmente faz com que a especificação de requisitos e a implementação do sistema se desajustem. Uma vez feitas as mudanças no sistema, as modificações no documento de requisitos podem ser esquecidas ou feitas de maneira não consistente com as mudanças no sistema.

Assim, resumindo, o processo de engenharia de requisitos inclui um estudo de viabilidade, levantamento e análise, especificação, validação e gerenciamento de requisitos. A análise é sempre um processo iterativo, que envolve a compreensão do domínio, assim como a coleta, classificação, estruturação, priorização e validação dos requisitos.



Por outro lado, diferentes *stakeholders* do sistema têm diferentes requisitos. Todos os sistemas complexos devem, portanto, ser analisados a partir de diferentes pontos de vista, os quais podem ser fontes ou "drenos" de dados. Assim, diferentes representações ou entidades do sistema estão fora do sistema e recebem serviços a partir deles. Isso faz com que fatores sociais e organizacionais tenham forte influência sobre os requisitos do sistema e possam determinar se o software será realmente utilizado ou não.

Finalmente, vale lembrar que a validação dos requisitos é o processo de verificar os requisitos quanto a sua validade, consistência, completude, realismo, e facilidade de verificação. As revisões de requisitos e a prototipação são as principais técnicas utilizadas para sua validação. Isso é importante porque mudanças sempre acontecem e mudanças organizacionais, técnicas e de negócios inevitavelmente levam a mudanças nos requisitos em um sistema de software, e o gerenciamento de requisitos é justamente o processo de gerenciar e controlar essas mudanças. Mas não apenas isso: o gerenciamento de requisitos inclui o planejamento do gerenciamento, em que são especificados os procedimentos e as políticas para gerenciamento de requisitos, e o gerenciamento de mudanças, em que as mudanças são analisadas e seu impacto é avaliado.

3. Projeto de software: requisitos

3.1 – Versão baseada no estado

Os modelos de objetos executáveis também são chamados Gráficos de Estado. Um gráfico de estado especifica o comportamento do sistema, a maneira como os objetos se comunicam e colaboram para alcançar algum objetivo. Os estados descrevem situações abstratas no ciclo de vida de um objeto. É bem fácil “ler” o código de um gráfico de estado, pois seus arcos são rotulados com condições de disparadores de listas de ações. Um disparador (Trigger) é um evento (por exemplo, modo = parar) ou solicitação de uma ação (por exemplo, alterar [termo]).

Uma ação é uma sequência de expressões que geram eventos, ou chamadas de operação, ou instruções em C++. Utilizando o conjunto de ferramentas Statemate MAG-NUM, os gráficos de estado podem ser executados e fornecer uma maneira de se estabelecerem protótipos rapidamente. Utilizando gráficos de estado, a descrição do sistema pode ser criada de forma modular, através do encapsulamento de informações.

Os detalhes do projeto de cada módulo são ocultos em camadas de gráficos de estado. Os estados na descrição de alto nível de um módulo podem ser decompostos em gráficos de estados mais detalhados. Os gráficos de estado de baixo nível revelam detalhes quer explicam o mecanismo subjacente do gráfico de estado de alto nível. O formalismo visual fornecido pelos gráficos de estado ajuda a compreensão e facilita uma descrição limpa e simples de hierarquias na estrutura operacional de um sistema.

3.2 – Resumo

Graças à disponibilidade de padrões para a especificação de requisitos de software, a estrutura funcional ERS é conhecida. O conteúdo de uma ERS será

direcionado pelo plano de projeto e pela interação com seus participantes. A ERS fornece descrição concisa de um aplicativo. Os objetivos, restrições e alternativas no desenvolvimento de requisitos são obtidos a partir da comunicação com os participantes. Os requisitos são desenvolvidos iterativamente com base nos comentários sobre os documentos da linha de base e na avaliação dos protótipos do sistema.

Sistemas como o Rhapsody, por exemplo, do i-logix, podem ser utilizados para gerar código a partir das descrições do gráfico de estado. Isso possibilita o estudo do comportamento das partes de um sistema durante a execução do protótipo. Se os recursos de geração de código não estiverem disponíveis, ainda assim será fácil codificar os incrementos selecionados do sistema descrito na ERS. Isso pode ser feito rapidamente, se for entendido que o protótipo é muito objetivo e que possibilita prova de conceito a partir de uma parte do sistema. Os protótipos rápidos são auxiliados pelo desenvolvimento incremental de requisitos. O projeto do protótipo codificado à mão deve refletir a descrição de um incremento.

Por fim, deve-se observar que pode ser útil utilizar a abordagem top-down. O segredo é iniciar o registro das ideias sobre a estrutura de alto nível do sistema. Em um gráfico de estado, por exemplo, alto nível pode ser decomposto em gráficos de estado mais detalhados, de mais baixo nível. As descrições de baixo nível ficam ocultas em um gráfico de estado de alto nível. Isso promove a compreensão dos principais recursos e funções do sistema. Os gráficos de estados de níveis mais baixos servem como explicações das funções principais do sistema.

REFERÊNCIAS:

DEVMIDIA. Introdução à Engenharia de Requisitos. 2008. Disponível em https://www.devmedia.com.br/introducao-a-engenharia-de-requisitos/8034. Acesso em 05 de fev. de 2023.

HULL, E. “Requirements Engineering”. USA: Springer, 2010.

IEEE (Institute of Electrical and Electronics Engineering). Guide to the Software Engineering Body of Knowledge - IEEE v3

MACHADO, F. N. “Análise e Gestão de Requisitos de Software”. São Paulo: Erica, 2011.

NOLETO, C. Engenharia de requisitos: quais as etapas e como funciona? 29 DE NOVEMBRO DE 2021. Disponível em https://blog.betrybe.com/tecnologia/engenharia-de-requisitos-tudo-sobre/. Acesso em 05 de fev. de 2023.

PETERS, J. F. Engenharia de Software. Rio de Janeiro: Campus, 2001.

PRESSMAN, R. S. Engenharia de Software: uma abordagem profissional. São Paulo: McGraw Hill, 2011.

ROVEDA, U. Engenharia de requisitos de software: o que é e como funciona. 01 de julho de 2022. Disponível em https://kenzie.com.br/blog/engenharia-de-requisitos-de-software/. Acesso em 05 de fev. de 2023.

SHUYTEMA, P. Design de games: uma abordagem prática. São Paulo: Cengage Learning, 2008.

SOMMERVILLE, I. Engenharia de Software. São Paulo: Pearson, 2011.