Certified Professional for Requirements Engineering

Nível Fundamental

Guia de Estudo

Martin Glinz
Hans van Loenhoud
Stefan Staal
Stan Bühne

Termos de uso

Agradecimentos

O conteúdo deste guia de estudo foi revisado por Rainer Grau, Karol Frühauf e Camille Salinesi. Tracey Duffy fez uma revisão em inglês. Stan Bühne e Stefan Sturm fizeram a edição final.

A versão 1.0.0 foi aprovada para liberação em 11 de novembro de 2020 pelo Conselho do IREB sob recomendação de Xavier Franch e Frank Houdek.

A versão 1.1.0 foi aprovada para liberação em 15 de agosto de 2022 pelo IREB ExCo. Traduzido do Inglês por Ana Moreira, Anselmo Peretto, Carlos André e Silva, George Fialkovitz, Guilherme Simões, Luciana Ribeiro, Martin Tornquist e Stênio Viveiros.

Agradecemos a todos pelo seu envolvimento.

Prefácio

Este Guia de Estudos fornece uma introdução à Engenharia de Requisitos baseada no syllabus 3.0 da Certified Professional for Requirements Engineering (CPRE) – Foundation Level de acordo com o IREB. Complementa o syllabus e aborda três grupos de leitores:

- *Alunos e profissionais* que desejam aprender sobre Engenharia de Requisitos e fazer o exame de certificação podem usar este guia de estudo como um livro complementar aos cursos de treinamento oferecidos por provedores de treinamento, bem como para autoestudo e preparação individual para o exame de certificação. Este guia de estudo também pode ser usado para atualizar o conhecimento existente sobre Engenharia de Requisitos, por exemplo, ao se preparar para um curso e exame CPRE de nível avançado.
- *Provedores de treinamento* que oferecem treinamentos no Nível Fundamental do CPRE podem usar este guia de estudo como complemento ao Syllabus para o desenvolvimento de seus materiais de treinamento ou como um texto de estudo para os participantes em seus treinamentos.
- *Profissionais* da indústria que desejam aplicar conceitos e conhecimentos comprovados da ER em seu trabalho prático encontrarão neste guia de estudo uma riqueza de informações úteis.

Este guia de estudo também fornece uma ligação entre o Syllabus, que lista e explica os objetivos de aprendizagem e a literatura sobre Engenharia de Requisitos. Cada capítulo vem com referências à literatura e dicas para leituras adicionais. A estrutura do guia de estudo corresponde à estrutura do Syllabus.

A terminologia usada neste guia de estudo é baseada no Glossário de Termos da Engenharia de Requisitos [Glin2020]. Recomendamos fazer o download deste glossário do site do IREB e usá-lo como referência de terminologia.

Você encontra mais informações sobre o programa de certificação CPRE, incluindo o syllabus, glossário, regulamentos do exame e exemplos de perguntas do exame no site do IREB em https://www.ireb.org.

Os autores e o IREB investiram uma quantidade significativa de tempo e esforço na preparação, revisão e publicação deste guia de estudo. Esperamos que goste de estudar por este guia de estudo. Se você detectar algum erro ou tiver sugestões para melhoria, por favor nos contate em info@ireb.org.

Gostaríamos de agradecer a todas as pessoas que contribuíram para a criação e publicação deste guia de estudo. Karol Frühauf, Rainer Grau e Camille Salinesi revisaram cuidadosamente o manuscrito e forneceram sugestões valiosas para melhorias. Tracey Duffy fez a revisão do inglês. Agradecemos também aos condutores no Conselho do IREB por este guia de estudo, Xavier Franch e Frank Houdek, por seus comentários e apoio. Stefan Sturm incentivou e deu apoio logístico. Agradecemos também nossos cônjuges e famílias por sua paciência e apoio.

Martin Glinz, Hans van Loenhoud, Stefan Staal e Stan Bühne

Novembro de 2020

Entendendo as caixas de texto deste guia de estudo

O guia de estudo inclui quatro caixas de texto com cores diferentes que complementam o texto explicativo.

São elas:

Definições (correspondentes ao Glossário [Glin2020])

Dicas

Exemplos

Expressões

Histórico das Versões

Versão	Data	Comentário	Autores
1.1.0	21 de setembro de 2023	Versão Inicial baseada no Guia de Estudo original do IREB em Inglês	Ana Moreira, Anselmo Peretto, Carlos André e Silva, George Fialkovitz, Guilherme Simões, Luciana Ribeiro, Martin Tornquist e Stênio Viveiros.
1.1.1	Janeiro de 2024	Novo design corporativo implementado	Stefan Sturm
1.2.0	15 de maio de 2024	Nós de junção ausentes adicionados à figura 3.11	Stefan Sturm

Conteúdo

1 Introdução e Visão Geral

Neste capítulo, você aprenderá o que é a Engenharia de Requisitos (ER) e o valor que ela traz.

1.1 Engenharia de Requisitos: O Quê

Desde o início de sua evolução, os humanos vêm construindo sistemas técnicos e organizacionais para *apoiá-los* na realização de tarefas ou objetivos. Com o surgimento da engenharia, eles também começaram a construir sistemas que *automatizam* tarefas humanas.

Sempre que decidimos construir um sistema para apoiar ou automatizar as tarefas humanas, temos que descobrir o que construir. Isso significa que eles têm que aprender sobre os desejos e as necessidades das pessoas ou organizações que usarão o sistema, se beneficiarão ou serão impactados. Em outras palavras, eles precisam conhecer os requisitos para este sistema. Os requisitos formam a base para qualquer desenvolvimento ou evolução de sistemas ou das suas partes. Os requisitos existem sempre, mesmo quando não são explicitamente capturados e documentados.

O termo *requisito* denota três conceitos [Glin2020]:

> **Definição 1.1. Requisito**
> 1. Uma necessidade percebida por um stakeholder
> 2. Uma capacidade ou propriedade que um *sistema* deve ter.
> 3. Uma representação documentada de uma necessidade, capacidade ou propriedade.

Uma coleção de requisitos sistematicamente representada – normalmente para um sistema – que satisfaça determinados critérios é chamada *especificação de requisitos*.

Distinguimos três tipos de requisitos:

- *Requisitos Funcionais* dizem respeito a um resultado ou comportamento que deve ser fornecido por uma função de um sistema. Isso inclui requisitos de dados ou de interação de um sistema com seu ambiente.
- *Requisitos de Qualidade* dizem respeito a questões de qualidade que não são cobertas por requisitos funcionais — por exemplo, desempenho, disponibilidade, segurança ou confiabilidade.
- *Restrições* são requisitos que limitam o espaço da solução além do que é necessário para atender aos requisitos funcionais e de qualidade.

Note que lidar com requisitos para projetos ou processos de desenvolvimento está fora do escopo deste guia de estudo.

A distinção entre requisitos funcionais, requisitos de qualidade e restrições nem sempre é simples. Uma maneira comprovada de diferenciá-los é perguntar de que o requisito trata: *se for sobre* resultados, comportamentos ou interações necessárias, temos um requisito funcional. Se é uma questão de qualidade que não está coberta por requisitos funcionais, temos um requisito de qualidade. Se a preocupação é limitar o espaço de solução, mas não é um requisito funcional nem de qualidade, temos uma restrição. A regra popular "O *que* o sistema deve fazer → requisito funcional versus *como* o sistema deve fazer → requisito de qualidade" frequentemente leva a classificações erradas, especialmente quando os requisitos são especificados detalhadamente ou quando os requisitos de qualidade são muito importantes.

Por exemplo, o requisito "O formulário de entrada do cliente deve conter campos para nome e sobrenome do cliente, ocupando até 32 caracteres por campo, sendo exibidos pelo menos 24 caracteres, ajustados à esquerda, com fonte sanserif 12 pts." é um requisito funcional embora contenha muitas informações sobre *o como*. Como outro exemplo, considere um sistema que processa os dados de medição produzidos pelo detector de uma acelerador de partículas de alta energia. Esses detectores produzem enormes quantidades de dados em tempo real. Se você perguntar a um físico "O que o sistema deve fazer?", uma das primeiras respostas seria provavelmente que o sistema deve ser capaz de lidar com o volume de dados produzido. No entanto, os requisitos relativos ao volume de dados ou velocidade de processamento são requisitos de qualidade [Glin2007] e não requisitos funcionais.

Quando as pessoas adotam uma abordagem sistemática e disciplinada para a especificação e gerenciamento de requisitos, chamamos isso de *Engenharia de Requisitos (ER)*. A seguinte definição de Engenharia de Requisitos também reflete o porquê de realizarmos ER.

> **Definição 1.2. Engenharia de Requisitos (ER)**
> A abordagem sistemática e disciplinada para a especificação e gerenciamento de requisitos com o objetivo de *compreender os desejos e necessidades dos stakeholders e minimizar o risco* de entrega de um sistema que não atende a esses desejos e necessidades.

O conceito de *stakeholder* [GlWi2007] é um princípio fundamental da Engenharia de Requisitos (vide capítulo 2).

Definição 1.3. Stakeholder:
Uma pessoa ou organização que influencia os requisitos de um sistema ou é impactada por esse sistema.

Observe que a influência também pode ser indireta. Por exemplo, alguns stakeholders podem ter que seguir as instruções emitidas por seus gerentes ou organizações.

Seguindo a definição no Glossário CPRE [Glin2020], neste guia de estudo usamos o termo *sistema* num sentido amplo:

Definição 1.4. Sistema:
1. Em geral: um princípio de ordenação e estruturação.
2. Na engenharia: um conjunto coerente e delimitável de elementos que, através de uma ação coordenada, alcancem algum objetivo.

Observe que um sistema pode compreender outros sistemas ou *componentes* como subsistemas. O propósito do sistema pode ser realizado ao:

- *Implantar* o sistema nos locais onde será usado
- Vender/licenciar o sistema a seus usuários como um *produto*
- Ter fornecedores que oferecem as capacidades do sistema aos usuários como *serviços*

Portanto, usamos o termo sistema como um termo abrangente que inclui produtos, serviços, aplicativos ou dispositivos.

1.2 Engenharia de Requisitos: O Porquê

Desenvolver sistemas (construir novos, bem como evoluir os existentes) é um empreendimento caro e de alto risco para todos os participantes. Ao mesmo tempo, sistemas relevantes são muito grandes para uma única pessoa entender intelectualmente. Por isso engenheiros criaram vários princípios e práticas no desenvolvimento de sistemas para lidar com este risco e complexidade intelectual. A Engenharia de Requisitos fornece os princípios e práticas na perspectiva dos requisitos.

Uma Engenharia de Requisitos (ER) adequada agrega *valor* [Glin2016], [Glin2008] ao processo de desenvolvimento de um sistema:

- A ER minimiza o risco de falhas ou modificações caras nas etapas posteriores do desenvolvimento. A detecção e correção precoce de requisitos errados ou ausentes é muito mais barata do que a correção de defeitos e retrabalhos causados por requisitos errados ou faltantes em estágios de desenvolvimento posteriores ou mesmo após a implantação do sistema.

- A ER alivia a complexidade intelectual para compreender o problema que um sistema deve resolver e refletir sobre possíveis soluções.
- A ER fornece uma base adequada para estimar o esforço e custo de desenvolvimento.
- A ER é um pré-requisito para testar o sistema adequadamente.

Os sintomas típicos de uma ER inadequada são requisitos em falta, pouco claros ou errados devido a:

- Equipes de desenvolvimento apressam-se a implementar um sistema devido à pressão do cronograma
- Problemas de comunicação entre as partes envolvidas, em particular entre *stakeholders* e desenvolvedores e entre os próprios *stakeholders*
- O pressuposto de que os requisitos são óbvios, o que é errado na maioria dos casos
- Pessoas realizando atividades da ER sem ter educação e habilidades adequadas

1.3 Engenharia de Requisitos: Onde

A Engenharia de Requisitos pode ser aplicada a requisitos de qualquer tipo de sistema. No entanto, hoje, ER aplica-se predominantemente a sistemas onde o software desempenha um papel fundamental. Esses sistemas consistem em componentes de software, elementos físicos (produtos técnicos, hardware de computação, dispositivos, sensores etc.) e elementos organizacionais (pessoas, cargos, processos de negócios, questões jurídicas e de conformidade etc.).

Sistemas que contêm software e componentes físicos são chamados *sistemas ciberfísicos*.

Sistemas que abrangem software, hardware, pessoas e aspectos organizacionais são chamados *sistemas sociotécnicos*.

Dependendo da perspectiva adotada, os requisitos ocorrem de várias formas:

Requisitos de Sistema descrevem como um sistema deve funcionar e se comportar – conforme observado na interface entre o sistema e seu ambiente – de modo a que o sistema satisfaça os desejos e necessidades dos seus stakeholders. No caso de sistemas de software puros, falamos de *requisitos de software*.

Requisitos de Stakeholder expressam os desejos e necessidades dos stakeholders que devem ser satisfeitos com a construção de um sistema, visto a partir da sua perspectiva.

Requisitos de Usuário são um subconjunto dos requisitos de stakeholder. Eles cobrem os desejos e necessidades dos usuários de um sistema.

Requisitos de Domínio especificam as propriedades de domínio de sistemas sociotécnicos ou ciberfísicos.

Requisitos de Negócio se concentram nos objetivos, metas e necessidades de negócios de uma organização que devem ser alcançadas através do emprego de um sistema (ou uma coleção de sistemas).

As formas de ocorrência apresentadas acima correspondem às definidas no padrão [ISO29148], com exceção dos requisitos de domínio. Pela sua importância, os tratamos como uma categoria própria. A função e a importância dos requisitos de domínio são discutidas na seção 2.2, Princípio 4.

1.4 Engenharia de Requisitos: Como

As principais tarefas da ER são a *elicitação* (capítulo 4), *documentação* (capítulo 3), *validação* (seção 4.4), e *gerenciamento* (capítulo 6) de requisitos. Ferramentas de suporte (capítulo 7) podem ajudar a realizar essas tarefas. A análise e a resolução de conflitos de requisitos são consideradas parte da elicitação.

No entanto, não existe um processo universal que descreva quando e como a ER deve ser realizada durante o desenvolvimento de um sistema. Para cada desenvolvimento de sistema que precisa de atividades de ER, um processo de ER adequado deve ser adaptado a partir de uma ampla gama de possibilidades. Os fatores que influenciam essa adaptação incluem, por exemplo:

- O processo geral de desenvolvimento do sistema – em particular, sequencial e orientado ao planejamento vs. iterativo e ágil
- O contexto do desenvolvimento, em particular, a relação entre os fornecedores, clientes e usuários do sistema
- A disponibilidade e capacidade dos stakeholders

Também há uma dependência mútua entre os produtos de trabalho de requisitos produzidos (vide seção 3.1) e o processo de ER escolhido. Mais detalhes são fornecidos no capítulo 5.

1.5 O Papel e as Tarefas de um Engenheiro de Requisitos

Na prática, poucas pessoas têm o cargo de *Engenheiro de Requisitos*. Consideramos que pessoas atuam no *papel* de um Engenheiro de Requisitos quando:

- Elicitam, documentam, validam e/ou gerenciam requisitos como parte de suas obrigações
- Tem profundo conhecimento de ER, o que lhes permite definir processos de ER, selecionar práticas de ER apropriadas e aplicá-las adequadamente
- São capazes de preencher a lacuna entre o problema e potenciais soluções

O papel do Engenheiro de Requisitos é parte de vários cargos definidos por organizações. Por exemplo, analistas de negócios, especialistas em aplicativos, Product Owners, engenheiros de sistemas e até mesmo desenvolvedores podem atuar no papel de um Engenheiro de Requisitos. Ter conhecimento e habilidades de ER também é útil para muitos outros profissionais – por exemplo, designers, testadores, arquitetos de sistema ou CTOs.

1.6 O que aprender sobre Engenharia de Requisitos

O conjunto de habilidades que um Engenheiro de Requisitos deve aprender consiste em vários elementos. Os elementos fundamentais são abordados nos capítulos subsequentes deste guia de estudo.

Além das habilidades técnicas e analíticas, um Engenheiro de Requisitos também precisa de habilidades interpessoais: a capacidade de ouvir, moderar, negociar, mediar e ter empatia pelos stakeholders e abertura às necessidades e ideias dos outros.

A ER é regida por um conjunto de princípios fundamentais que se aplicam a todas as tarefas, atividades e práticas da ER. Esses princípios são apresentados no Capítulo 2.

Os requisitos podem ser documentados de várias formas. Vários produtos de trabalho podem ser criados em diferentes níveis de maturidade e detalhe, de informais e temporários aos muito detalhados e estruturados que seguem regras estritas de representação. É importante selecionar produtos de trabalho e formas de documentação que sejam adequadas para cada situação em questão e criá-los corretamente. Produtos de trabalho e práticas de documentação são apresentados no Capítulo 3.

Os requisitos podem ser elaborados (ou seja, elicitados e validados) através de várias práticas. O Engenheiro de Requisitos deve ser capaz de selecionar as práticas mais adequadas para cada situação e aplicá-las de forma adequada. As práticas de elaboração são apresentadas no Capítulo 4.

Compreender os possíveis processos e estruturas de trabalho permite aos Engenheiros de Requisitos definir uma forma de trabalhar que se adapte às necessidades específicas da abordagem de desenvolvimento do sistema em questão. Processos e estruturas de trabalho são apresentados no Capítulo 5.

Requisitos existentes podem ser gerenciados através de várias práticas. Engenheiros de Requisitos devem ser capazes de entender quais práticas de gerenciamento de requisitos os apoiam em quais tarefas. As práticas de gerenciamento são apresentadas no Capítulo 6.

Ferramentas tornam a ER mais eficiente. Os Engenheiros de Requisitos precisam saber como as ferramentas da ER podem apoiá-los e como selecionar uma ferramenta adequada para sua situação. O suporte por ferramentas é discutido brevemente no Capítulo 7.

1.7 Leitura adicional

A terminologia da ER usada neste guia de estudo é definida no Glossário CPRE de Termos de Engenharia de Requisitos [Glin2020]. Glinz e Wieringa [GlWi2007] explicam a noção de stakeholder. Lawrence, Wiegers e Ebert [LaWE2001] discutem brevemente os riscos e armadilhas da ER.

Gause e Weinberg [GaWe1989] escreveram um dos primeiros livros didáticos sobre ER, que ainda vale a pena olhar. Pohl [Pohl2010], Robertson e Robertson [RoRo2012] e Wiegers e Beatty [WiBe2013] são livros didáticos populares sobre ER. As notas do curso de Glinz [Glin2019] fornecem uma introdução baseada em slides para a ER. O livro didático de van

Lamsweerde [vLam2009] apresenta uma abordagem da ER orientada a objetivos. Jackson [Jack1995] contribui com uma coleção criteriosa de ensaios sobre requisitos de software.

Esteja ciente de que o livro oficial para a versão 2.2 do Syllabus CPRE-FL [PoRu2015] não está mais totalmente alinhado com a versão 3.0 do Syllabus CPRE-FL, no qual este guia de estudo se baseia. No entanto, este livro ainda fornece uma introdução concisa ao ER e será atualizado em breve.

Existem também livros didáticos em outros idiomas além do inglês. Como exemplo, Badreau e Boulanger [BaBo2014] escreveram um livro de ER em francês. Os livros de Ebert [Eber2014] e Rupp [Rupp2014] são livros populares da ER escritos em Alemão.

2 Princípios Fundamentais da Engenharia de Requisitos

Neste capítulo, você aprenderá sobre nove princípios fundamentais da Engenharia de Requisitos (ER).

2.1 Visão Geral dos Princípios

A ER é regida por um conjunto de princípios fundamentais que se aplicam a todas as suas tarefas, atividades e práticas. Uma *tarefa* é um pedaço coerente de trabalho a ser feito (por exemplo, elicitar requisitos). Uma *atividade* é uma ação ou um conjunto de ações que uma pessoa ou grupo executa para realizar uma tarefa (por exemplo, identificar os stakeholders ao elicitar os requisitos). Uma *prática* é uma forma comprovada de como realizar certos tipos de tarefas ou atividades (p. ex., utilizando entrevistas para obter requisitos dos stakeholders).

Os princípios listados em Tabela 2.1 formam a base para as práticas apresentadas nos capítulos seguintes deste guia de estudo.

Tabela 2.1 Nove Princípios Fundamentais da Engenharia de Requisitos

1.	*Orientação para o valor*: Os requisitos são um meio para um fim, não um fim em si mesmo
2.	*Stakeholders*: ER é sobre satisfazer os desejos e necessidades dos stakeholders
3.	*Entendimento comum*: O desenvolvimento bem-sucedido de sistemas é impossível sem uma base de compreensão comum
4.	*Contexto*: Sistemas não podem ser compreendidos isoladamente
5.	*Problema – Requisito – Solução*: Uma trinca inevitavelmente entrelaçada
6.	*Validação*: Requisitos não validados são inúteis
7.	*Evolução*: A mudança de requisitos não é um acidente, mas o caso normal
8.	*Inovação*: Mais do mesmo não é mais o suficiente
9.	*Trabalho sistemático e disciplinado*: Não podemos prescindir deles na ER

2.2 Os Princípios Explicados

2.2.1 Princípio 1 - Orientação para o valor: Os requisitos são um meio para um fim, não um fim em si mesmo

O ato de escrever requisitos não é um objetivo por si só. Os requisitos são úteis – e o esforço investido na Engenharia de Requisitos se justifica – se eles agregarem *valor* [Glin2016], [Glin2008], cf. Seção 1.2. Definimos o valor de um requisito como sendo seu *benefício* menos seu *custo*. O benefício de um requisito é o grau em que ele contribui para construir sistemas bem-sucedidos (isto é, sistemas que satisfaçam os desejos e necessidades de seus stakeholders) e para reduzir o risco de falha e de retrabalho dispendioso no desenvolvimento de sistemas. O custo de um requisito equivale ao custo para elicitá-lo, validá-lo, documentá-lo e gerenciá-lo.

A redução do risco de retrabalho durante o desenvolvimento é uma parte constituinte do benefício de um requisito bem elaborado. Detectar e corrigir um requisito ausente ou errado durante a implementação ou quando o sistema já estiver em operação pode facilmente custar uma ou duas ordens de grandeza a mais do que especificar corretamente esse requisito desde o início. Consequentemente, uma parte significativa do benefício dos requisitos vem dos custos economizados durante a implementação e operação de um sistema.

Em outras palavras, os benefícios da ER são muitas vezes não imediatos, enquanto os custos são imediatos. Isto deve ser levado em conta ao se montar um novo projeto. A redução dos custos a curto prazo, gastando menos com a ER, tem um preço: aumenta consideravelmente o risco de retrabalho caro em etapas posteriores do projeto.

O *valor da Engenharia de Requisitos* pode ser considerado como o valor cumulativo dos requisitos especificados. Como os clientes normalmente pagam pelos sistemas a serem implementados, mas não pelos requisitos necessários para fazer isso, o valor econômico da ER é, em sua maioria, indireto. Este efeito é reforçado pelo fato de que o benefício dos requisitos que decorrem da redução dos custos de retrabalho é indireto: ele economiza custos durante a implementação e operação.

> O valor econômico da Engenharia de Requisitos é principalmente indireto; a ER como tal, apenas custo.

Para otimizar o valor de um requisito, os Engenheiros de Requisito têm que encontrar um equilíbrio adequado entre o benefício e o custo de um requisito. Por exemplo, elicitar e documentar a necessidade de um stakeholder como um requisito facilita a comunicação desta necessidade entre todas as partes envolvidas. Isto aumenta a probabilidade de que o sistema a ser construído eventualmente satisfaça esta necessidade, o que constitui um benefício. Quanto menos ambíguo e mais preciso for o requisito, maior será seu benefício, pois isso reduz o risco de retrabalho dispendioso devido à má interpretação dos requisitos

pelos arquitetos e equipes de desenvolvimento do sistema. Por outro lado, o aumento do grau de clareza e precisão de um requisito também aumenta o custo envolvido na sua elaboração e documentação.

Na verdade, a quantidade de ER necessária para atingir requisitos com valor ideal depende de numerosos fatores dados pela situação específica na qual os requisitos estão sendo criados e utilizados. Obviamente, o risco de construir um sistema que eventualmente não satisfaça os desejos e necessidades de seus stakeholders, o que pode resultar em falha ou retrabalho dispendioso, *é a força motriz* que determina a quantidade de ER necessária. Antes de tudo, a criticidade de cada requisito deve ser avaliada em termos da importância dos stakeholders que formularam o requisito (ver Princípio 2) e o impacto da falta do requisito (Figura 2.1).

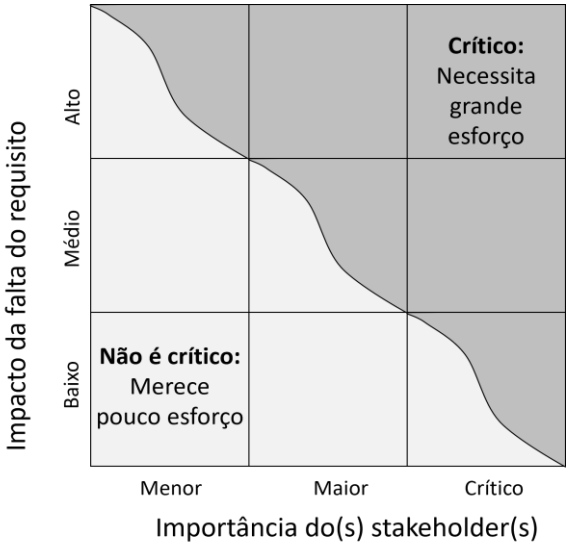


Figura 2.1 Avaliando a criticidade de um requisito [Glin2008]

Além disso, os seguintes fatores de influência devem ser considerados:

- Esforço necessário para especificar o requisito
- O caráter distinto do requisito (o quanto ele contribui para sucesso global do sistema)
- Grau de entendimento compartilhado entre stakeholders e desenvolvedores e entre si
- Existência de sistemas de referência (que podem servir de especificação por exemplo)
- Duração do ciclo de feedback (o tempo entre a criação de um requisito errado e a detecção do erro)
- Tipo de relação cliente-fornecedor
- Conformidade regulatória necessária

Resumimos esta questão em duas regras práticas:

- A quantidade ideal de ER a ser investida depende da situação específica e é determinada pela influência de muitos fatores.

- O esforço investido em ER deve ser inversamente proporcional ao risco que você está disposto a assumir.

2.2.2 Princípio 2 - stakeholders: ER é sobre satisfazer os desejos e necessidades dos stakeholders

O objetivo final da construção de um sistema é que o sistema, quando utilizado, solucione problemas que seus usuários precisam resolver e satisfaça as expectativas de outras pessoas – por exemplo, aqueles que encomendaram e pagaram pelo sistema, ou aqueles que são responsáveis pela segurança na organização que utiliza o sistema. Portanto, temos que descobrir as necessidades e expectativas das pessoas que têm um interesse no sistema, os *stakeholders* do sistema [GlWi2007]. Os objetivos centrais da ER são *compreender os desejos e necessidades dos stakeholders* e *minimizar o risco* de entrega de um sistema que não atende a esses desejos e necessidades; ver Definição 1.2 na Seção 1.2.

Cada stakeholder tem um papel no contexto do sistema a ser construído, por exemplo, desenvolvedor, cliente, operador ou *regulador*. Dependendo do processo de ER utilizado, os desenvolvedores de um sistema também podem ser stakeholders. Este é frequentemente o caso no desenvolvimento Ágil e orientado ao mercado. Um stakeholder também pode ter mais de um papel simultaneamente. Para cada papel relevante de stakeholder, as pessoas adequadas que atuam neste papel devem ser selecionadas como representantes.

Para papéis de stakeholders com muitos indivíduos ou quando os indivíduos são desconhecidos, *personas* (personagens fictícios que representam um grupo de usuários com características semelhantes) podem ser definidos como um substituto. Para sistemas que já estão em uso, os usuários que fornecem feedback sobre o sistema ou pedem por novas características também devem ser considerados como stakeholders.

Faz sentido classificar os stakeholders em três categorias com relação ao seu grau de influência sobre o sucesso do sistema:

- *Alta:* não considerar estes stakeholders resultará em problemas graves e provavelmente fará com que o sistema falhe ou se torne inútil.
- *Média:* não considerar esses stakeholders terá um impacto adverso no sucesso do sistema, mas não fará com que falhe.
- *Baixa:* não considerar esses stakeholders terá nenhuma ou pouca influência sobre o sucesso do sistema.

Esta classificação é útil ao avaliar a criticidade de um requisito (ver Figura 2.1) e ao negociar conflitos entre os stakeholders (ver abaixo).

Não é suficiente considerar apenas os requisitos dos usuários finais ou dos clientes. Fazer isso significaria que poderíamos perder requisitos essenciais de outros stakeholders, o que pode facilmente levar a projetos de desenvolvimento que falham ou ultrapassam seus orçamentos e prazos.

Envolver as pessoas certas nos papéis relevantes de stakeholder é crucial para o sucesso da ER.

As práticas para identificar, priorizar e trabalhar com os stakeholders são discutidas no Capítulo 4.

Os stakeholders em diferentes papéis têm naturalmente pontos de vista diferentes [NuKF2003] de um sistema a ser desenvolvido. Por exemplo, os usuários normalmente querem um sistema que suporte suas tarefas de forma otimizada, os gerentes que encomendam o sistema querem obtê-lo a um custo razoável e o chefe de segurança da organização se preocupa principalmente com a segurança do sistema. Mesmo stakeholders no mesmo papel podem ter necessidades diferentes. Por exemplo, no grupo de usuários finais, usuários eventuais têm requisitos de interface de usuário que podem diferir fortemente das dos usuários profissionais.

Como consequência, não é suficiente apenas coletar os requisitos dos stakeholders. É vital identificar inconsistências e conflitos entre os requisitos dos diferentes stakeholders e resolvê-los, seja por meio da busca de um consenso, seja por imposição ou da especificação de variantes do sistema para stakeholders que, de fato, têm necessidades diferentes; ver Seção 4.3.

2.2.3 Princípio 3 - Entendimento comum: O desenvolvimento bem-sucedido de sistemas é impossível sem uma base de conhecimento comum

O desenvolvimento de sistemas, incluindo a ER, é um esforço de várias pessoas. Para que tal empreendimento seja um sucesso, as pessoas envolvidas precisam de um *entendimento compartilhado* do problema e dos requisitos que dele decorrem [GlFr2015].

A ER cria, fomenta e assegura o entendimento comum entre e dentre as partes envolvidas: stakeholders, Engenheiros de Requisitos e desenvolvedores. Fazemos distinção entre duas formas de entendimento compartilhado:

- O entendimento *explícito e compartilhado* é alcançado através de requisitos cuidadosamente elaborados, documentados e acordados. Este é o objetivo principal da ER nos processos orientados por planejamento.
- *Entendimento comum implícito* é baseada no conhecimento comum de necessidades, visões, contexto etc. Em uma ER Ágil, quando os requisitos não são totalmente especificados por escrito, a confiança no entendimento implícito compartilhado é fundamental.

Tanto o entendimento compartilhado implícito quanto o explícito podem ser *falsos*, o que significa que as pessoas acreditam que têm um entendimento compartilhado de uma questão, mas na verdade interpretam esta questão de maneiras diferentes. Portanto, nunca podemos confiar cegamente no entendimento compartilhado. Em vez disso, a tarefa da ER

é criar e fomentar o entendimento compartilhado e também assegurá-lo – isto é, avaliar se existe um verdadeiro entendimento compartilhado. Para limitar o esforço envolvido, é vital concentrar-se no entendimento compartilhado de coisas *relevantes* – ou seja, aqueles aspectos que se encontram dentro dos limites do contexto de um sistema (cf. Princípio 4).

Mesmo com um perfeito entendimento compartilhado, requisitos importantes ainda podem não ser atendidos porque ninguém os considerou. Figura 2.2 ilustra diferentes situações de entendimento compartilhado com um simples exemplo de um casal que quer instalar um balanço em seu jardim para seus filhos [Glin2019]. O post-it no meio simboliza uma especificação escrita.

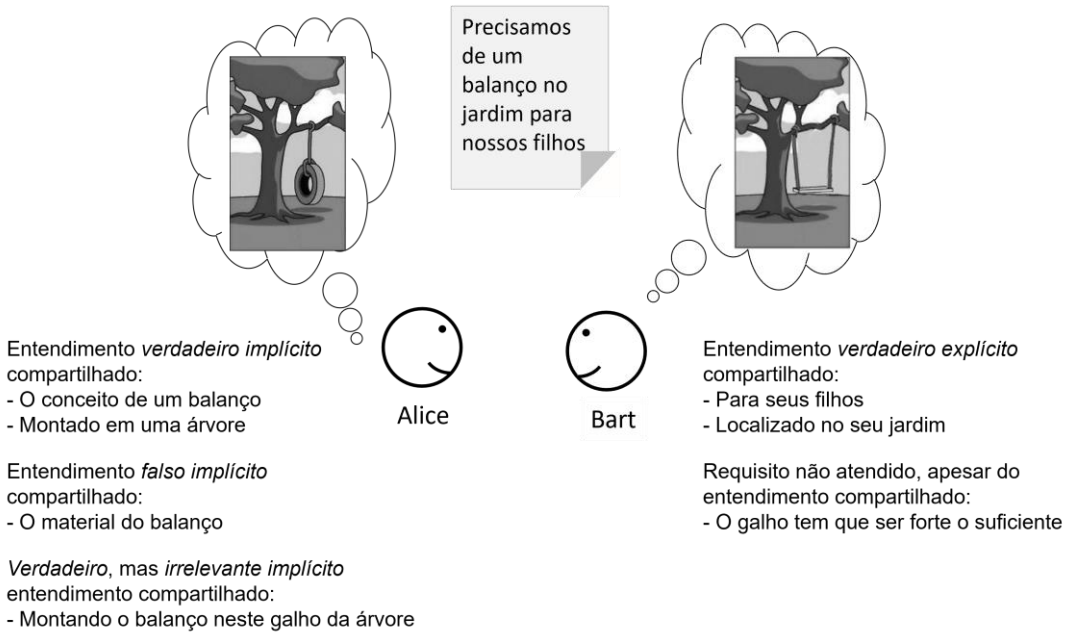


Figura 2.2 Diferentes situações de entendimento compartilhado - ilustrados com um exemplo de um casal que quer instalar um balanço para seus filhos

As práticas comprovadas para *alcançar* o entendimento comum incluem a criação de glossários (Seção 3.5), protótipos (Seção 3.7), ou o uso de um sistema existente como ponto de referência.

O principal meio para *avaliar o* verdadeiro entendimento explícito compartilhado na ER é validar completamente todos os requisitos especificados (cf. Princípio 6 e Seção 4.4). As práticas para avaliar o entendimento comum implícito incluem fornecer exemplos de resultados esperados, construir protótipos ou estimar o custo da implementação de um requisito. A prática mais importante para reduzir o impacto do falso entendimento compartilhado é a utilização de um processo com ciclos de feedback curtos (Capítulo 5).

Há fatores que constituem facilitadores ou obstáculos de entendimento compartilhado. Por exemplo, os facilitadores são:

- Conhecimento do domínio
- Normas específicas de domínio

- Colaboração anterior bem-sucedida
- Existência de sistemas de referência conhecidos por todas as pessoas envolvidas
- Cultura e valores compartilhados
- Confiança mútua informada (não cega!)

Os obstáculos são:

- Distância geográfica
- Relação fornecedor-cliente guiada pela desconfiança mútua
- Terceirização
- Restrições regulatórias
- Equipes grandes e diversificadas
- Alta rotatividade entre as pessoas envolvidas

Quanto menor a probabilidade e o impacto de falso entendimento compartilhado e quanto melhor a relação entre facilitadores e obstáculos, mais a ER pode se basear em entendimento implícito compartilhado. Por outro lado, quanto menos facilitadores e mais obstáculos a entendimento compartilhado tivermos e quanto maior o risco e o impacto de falso entendimento compartilhado de um requisito, mais tais requisitos precisam ser detalhados e validados explicitamente.

2.2.4 Princípio 4 - Contexto: Sistemas não podem ser entendidos isoladamente

Os requisitos nunca vêm isolados. Eles se referem a *sistemas* que estão embutidos em um *contexto*. Enquanto o termo *contexto* em geral denota a rede de pensamentos e significados necessários para compreender fenômenos ou afirmações, ele tem um significado especial na ER.

Definição 2.1. Contexto:
A parte do ambiente de um sistema sendo relevante para a compreensão do sistema e de seus requisitos.

O contexto de um sistema é delimitado pelo limite do sistema e pelo limite do contexto [Pohl2010] (ver Figura 2.3).

Definição 2.2. Limite do Contexto:
A fronteira entre o contexto de um sistema e as partes do domínio de aplicação que são irrelevantes para o sistema e seus requisitos.

O *limite do contexto* separa a parte relevante do ambiente de um sistema a ser desenvolvido da parte irrelevante – ou seja, a parte que não influencia o sistema a ser desenvolvido e, portanto, não tem que ser considerada durante a Engenharia de Requisitos.

Definição 2.3. Limite do sistema:
É a fronteira entre o sistema e o seu contexto adjacente.

O *limite do sistema* delimita o sistema como ele será após sua implementação e implantação. Inicialmente, o limite do sistema pode não ser claro, podendo mudar com o tempo. Esclarecer o limite do sistema e definir as interfaces externas entre o sistema e os elementos de contexto com os quais interage são tarefas genuínas da ER.

O limite do sistema frequentemente coincide com o *escopo* do desenvolvimento de um sistema.

Definição 2.4. Escopo:
A gama de coisas que podem ser modificadas e projetadas ao desenvolver um sistema.

Algumas vezes, no entanto, o limite do sistema e seu escopo não coincidem (ver Figura 2.3). Pode haver componentes dentro do limite do sistema que precisam ser reutilizados como estão (ou seja, não podem ser modificados ou redesenhados), o que significa que eles estão fora do escopo. Por outro lado, pode haver coisas no contexto do sistema que podem ser redesenhadas quando o sistema é desenvolvido, o que significa que elas estão no escopo.

Como as interfaces externas residem no limite do sistema, a ER deve determinar quais dessas interfaces estão no escopo (ou seja, podem ser moldadas e projetadas no processo de desenvolvimento) e quais estão fora do escopo.

Não é suficiente considerar apenas os requisitos dentro do limite do sistema.

Primeiro, quando o escopo inclui partes do contexto do sistema, como mostrado em Figura 2.3, mudanças de contexto dentro do escopo podem impactar os requisitos do sistema. Por exemplo, quando um processo negocial é parcialmente automatizado por um sistema, pode ser útil adaptar o processo a fim de simplificar sua automação. Obviamente, tal adaptação afeta os requisitos do sistema.

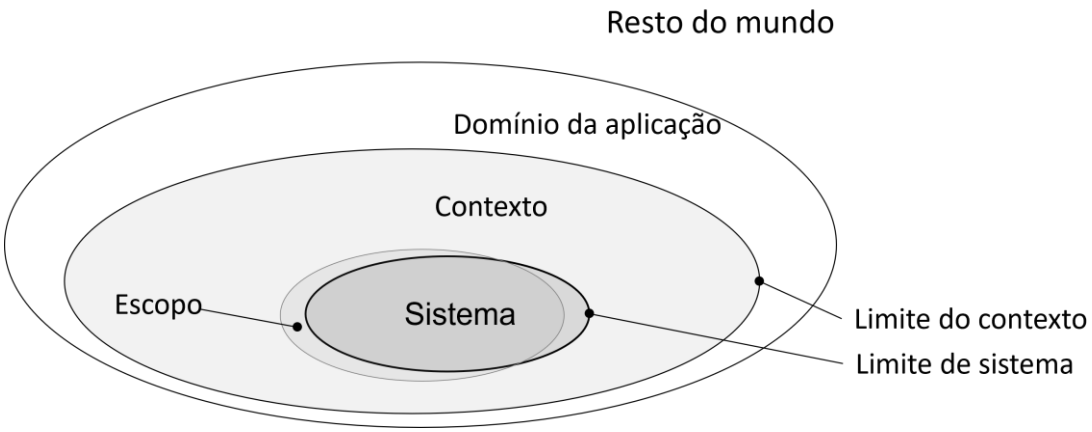


Figura 2.3 Sistema, contexto e escopo

Em segundo lugar, pode haver fenômenos do mundo real no contexto do sistema que um sistema deve monitorar ou controlar. Os requisitos para tais fenômenos devem ser declarados como requisitos de domínio e devem ser mapeados adequadamente aos requisitos do sistema. Por exemplo, em um carro equipado com uma caixa de câmbio automática, há o requisito de que a posição de estacionamento só possa ser engatada quando o carro não estiver em movimento. No contexto de um sistema de software que controla a troca de marchas, este é um requisito de domínio. Para satisfazer este requisito, o controlador precisa saber se o carro está ou não em movimento. Entretanto, o controlador não pode sentir este fenômeno diretamente. Portanto, o fenômeno do mundo real "carro não está em movimento" deve ser mapeado para um fenômeno que o sistema de controle pode detectar – por exemplo, a entrada de um sensor que cria pulsos quando uma roda do carro está girando. O requisito de domínio relativo ao acionamento da posição de estacionamento é então mapeado para um requisito de sistema como "O sistema de controle da caixa de câmbio deve permitir o acionamento da posição de estacionamento somente se não forem recebidos pulsos dos sensores de rotação da roda".

Em terceiro lugar, pode haver requisitos que não podem ser satisfeitos por qualquer implementação do sistema, a menos que certos *requisitos de domínio* e *suposições de domínio* no contexto do sistema sejam válidos. As suposições de domínio são suposições sobre fenômenos do mundo real no contexto de um sistema. Por exemplo, considere um sistema de controle de tráfego aéreo (SCTA). O requisito "R1: O SCTA deve manter posições precisas para todas as aeronaves controladas pelo sistema" é um requisito importante do sistema. Entretanto, este requisito só pode ser atendido se o radar no contexto do SCTA satisfizer os requisitos de identificar corretamente todas as aeronaves no espaço aéreo controlado pelo radar e determinar corretamente sua posição. Por sua vez, estes requisitos só podem ser satisfeitos se todas as aeronaves avistadas pelo radar responderem adequadamente aos sinais de interrogação enviados pelo radar.

Além disso, o requisito R1 só pode ser cumprido se certas suposições de domínio no contexto do SCTA, por exemplo, que o radar não está bloqueado por um ataque malicioso e que nenhuma aeronave está voando a uma altitude inferior à que o radar pode detectar.

A ER vai além de considerar os requisitos dentro do limite do sistema e definir as interfaces externas no limite do sistema. A ER também deve lidar com fenômenos no contexto do sistema.

Consequentemente, a ER também deve considerar questões no contexto do sistema:

- Se mudanças no contexto podem ocorrer, como elas afetam os requisitos para o sistema?
- Quais requisitos no contexto do mundo real são relevantes para o sistema a ser desenvolvido?
- Como esses requisitos do mundo real podem ser mapeados adequadamente aos requisitos do sistema?
- Quais suposições sobre o contexto devem ser asseguradas para que o sistema funcione corretamente e os requisitos do mundo real sejam atendidos?

2.2.5 Princípio 5 - Problema, requisito e solução: Um trio inevitavelmente entrelaçado

Os problemas, suas soluções e requisitos estão estreita e inevitavelmente entrelaçados [SwBa1982]. Toda situação em que as pessoas não estão satisfeitas com a maneira como estão fazendo as coisas pode ser considerada como a ocorrência de um *problema*. A fim de eliminar esse problema, um sistema sociotécnico pode ser desenvolvido e implantado. *Os requisitos* para esse sistema devem ser capturados a fim de tornar o sistema uma *solução* eficaz para o problema. A especificação de requisitos não faz sentido se não houver problema a resolver ou se não for desenvolvida uma solução. Também não faz sentido desenvolver uma solução que esteja procurando um problema para resolver ou requisito para satisfazer.

É importante notar que problemas, requisitos e soluções não ocorrem necessariamente nesta ordem. Por exemplo, ao projetar um sistema inovador, as ideias de solução criam necessidades do desenvolvedor que devem ser trabalhadas como requisitos e implementadas em uma solução real.

Problemas, requisitos e soluções podem estar entrelaçados de muitas maneiras:

- Entrelaçamento hierárquico: ao desenvolver grandes sistemas com uma hierarquia multinível de subsistemas e componentes, os requisitos de alto nível levam a decisões de desenho de alto nível, que por sua vez informam os requisitos de nível inferior que levam a decisões de desenho de nível inferior etc.
- Viabilidade técnica: especificar requisitos não viáveis é um desperdício de esforço; no entanto, só é possível avaliar a viabilidade de um requisito ao explorar soluções técnicas.
- Validação: os protótipos, que são um meio poderoso para validar os requisitos, constituem soluções parciais do problema.

- Viés de solução: diferentes stakeholders podem conceber soluções diferentes para um determinado problema, com a consequência de especificarem requisitos diferentes e conflitantes para o mesmo problema.

O entrelaçamento de problemas, requisitos e soluções também tem consequências para o processo de desenvolvimento de um sistema:

- Raramente é possível separar rigorosamente a ER das atividades de projeto e implementação do sistema. Assim, os rigorosos processos de desenvolvimento em cascata não funcionam bem.
- No entanto, os Engenheiros de Requisitos procuram separar problemas, requisitos e soluções uns dos outros, tanto quanto possível, ao pensar, comunicar e documentar. Esta *separação de preocupações* torna mais fácil lidar com as tarefas da ER.

Apesar do inevitável entrelaçamento de problemas, requisitos e soluções, os Engenheiros de Requisitos se esforçam para separar as preocupações com requisitos das preocupações com soluções ao pensar, comunicar e documentar.

2.2.6 Princípio 6 - Validação: Requisitos não validados são inúteis

Quando um sistema é desenvolvido, o sistema final implantado deve satisfazer os desejos e necessidades dos stakeholders. No entanto, realizar esta verificação no final do desenvolvimento é muito arriscado. A fim de controlar desde o início o risco de stakeholders insatisfeitos, a validação dos requisitos deve começar já durante a ER (veja Figura 2.4).

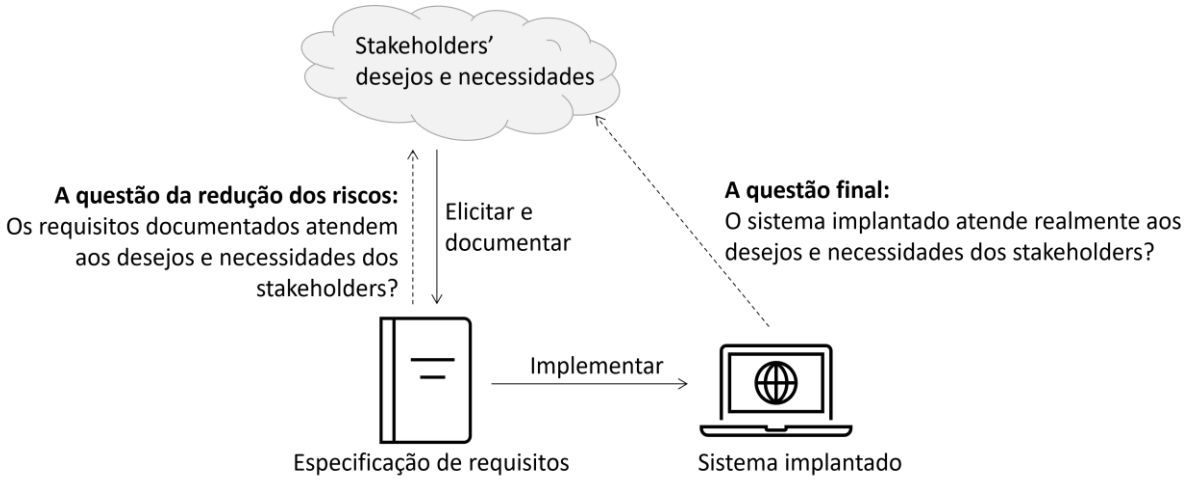


Figura 2.4 Validação [Glin2019]

> **Definição 2.5. Validação:**
> O processo de confirmação de que um item (um sistema, um produto de trabalho ou uma parte dele) atende às necessidades de seus stakeholders.

Na ER, a *validação* é o processo de confirmar que os requisitos documentados correspondem às necessidades dos stakeholders; em outras palavras, confirmar se os requisitos corretos foram especificados.

A validação é uma atividade central na ER: não há especificação sem validação.

Ao validar os requisitos, temos que verificar se:

- Foi alcançado um acordo sobre os requisitos entre os stakeholders (conflitos resolvidos, prioridades estabelecidas)
- Os desejos e necessidades dos stakeholders são devidamente cobertos pelos requisitos
- As suposições de domínio (ver Princípio 4 acima) são razoáveis – ou seja, podemos esperar que essas suposições sejam asseguradas quando o sistema for implantado e operado

As práticas para validação dos requisitos são discutidas em 4.4.

2.2.7 Princípio 7 - Evolução: A mudança de requisitos não é um acidente, mas o caso normal

Todo sistema técnico está sujeito a evolução. As necessidades, os negócios e as capacidades mudam continuamente. Como consequência natural, os requisitos para sistemas que devem satisfazer as necessidades, apoiar as empresas e usar as capacidades técnicas também mudarão. Caso contrário, tais sistemas e seus requisitos perdem progressivamente seu valor e acabam se tornando inúteis.

Um requisito pode mudar enquanto os Engenheiros de Requisitos ainda estão elicitando outros requisitos, quando o sistema está em implementação, ou quando ele está sendo implantado e sendo usado.

Há muitas razões que levam a pedidos para alterar um requisito ou um conjunto de requisitos para um sistema, por exemplo:

- Processos de negócio alterados
- Concorrentes que lançam novos produtos ou serviços
- Clientes que mudam suas prioridades ou opiniões

- Mudanças na tecnologia
- Feedback de usuários do sistema pedindo recursos novos ou alterados
- Detecção de erros nos requisitos ou detecção de suposições de domínio defeituosas

Além disso, os requisitos podem mudar devido ao feedback dos stakeholders ao validá-los, devido à detecção de falhas em requisitos elicitados anteriormente, ou devido a necessidades alteradas.

Como consequência, os Engenheiros de Requisitos devem buscar dois objetivos contraditórios:

- Permitir que os requisitos sejam alterados, pois tentar ignorar a evolução dos mesmos seria inútil.
- Manter os requisitos estáveis, porque sem alguma estabilidade nos requisitos, o custo da mudança pode se tornar proibitivamente alto. Além disso, as equipes de desenvolvimento não podem desenvolver sistematicamente se os requisitos mudarem diariamente.

Os Engenheiros de Requisitos precisam gerenciar a evolução dos requisitos. Caso contrário, a evolução vai gerenciá-los.

Os processos de mudança para requisitos que atendem a ambos os objetivos são discutidos na Seção 6.7.

2.2.8 Princípio 8 - Inovação: Mais do mesmo não é mais o suficiente

Enquanto a ER está preocupada em satisfazer os desejos e necessidades dos stakeholders, os Engenheiros de Requisitos que apenas desempenham o papel de 'gravador de voz' dos mesmos, especificando exatamente o que os stakeholders lhes pedem, estão fazendo o trabalho errado. Dar aos stakeholders exatamente o que eles querem, significa perder a oportunidade de fazer as coisas melhor do que antes.

Imagine, por exemplo, o seguinte cenário. Uma companhia de seguros quer renovar o sistema de relatórios de acompanhamento para seus corretores. O relatório mais utilizado é uma tabela com 18 colunas, que é cerca do dobro da largura da tela quando exibida nos computadores portáteis dos corretores. A visualização deste relatório requer, portanto, muita rolagem de tela. Os stakeholders, portanto, querem uma função zoom para este relatório, usando os botões mais e menos na tela. Nesta situação, bons Engenheiros de Requisitos não registrarão isto apenas como um requisito. Ao invés disso, eles começarão a fazer perguntas. Acontece que a empresa vai substituir os laptops dos corretores por 'tablets'. Portanto, implementar gestos com dois dedos em vez dos botões tornará o zoom muito mais fácil. Além disso, acontece que três colunas do relatório podem ser eliminadas com uma ligeira mudança nas regras do relatório, o que a empresa concorda em fazer. Além disso, apenas seis colunas do relatório são sempre necessárias; as colunas restantes são utilizadas apenas em casos especiais.

Levando isto em consideração, os Engenheiros de Requisitos sugerem que os stakeholders solicitem que (1) o relatório apresente as mesmas informações que no sistema atual, menos o conteúdo das três colunas eliminadas; (2) quando o relatório é aberto, apenas as seis colunas importantes são exibidas em largura total, enquanto as outras colunas são colapsadas até a largura mínima; e (3) que os corretores possam expandir uma coluna colapsada tocando em seu cabeçalho (e colapsá-la novamente com outra toque).

Desta forma, os corretores terão um sistema que não simplesmente acrescenta uma alternativa para a visualização de um relatório superdimensionado. Em vez disso, o sistema resolverá o problema dos corretores com um recurso inovador de filtragem de informações e também contará com um meio intuitivo de fazer zoom.

É assim que surge a inovação. Bons Engenheiros de Requisitos são sensíveis à inovação: eles se esforçam não apenas para satisfazer os stakeholders, mas para fazê-los felizes, entusiasmados ou se sentirem seguros [KSTT1984]. Ao mesmo tempo, eles evitam a armadilha de acreditar que sabem tudo melhor do que os stakeholders.

> Bons Engenheiros de Requisitos vão além do que seus stakeholders lhes pedem.

Em pequena escala, a ER dá forma a sistemas inovadores, esforçando-se por novas e excitantes funcionalidades e simplicidade de uso. Além disso, os Engenheiros de Requisitos também precisam visualizar 'o todo', explorando com os stakeholders se existem formas disruptivas de fazer as coisas, levando à inovação maciça [MaGR2004].

A seção 4.2 discute várias técnicas para promover a inovação na ER.

2.2.9 Princípio 9 - Trabalho sistemático e disciplinado: Não podemos fazer sem a ER

A ER não é uma arte, mas uma disciplina, o que exige que a ER seja realizada de forma sistemática e disciplinada. Independentemente dos processos utilizados para desenvolver um sistema, precisam empregar processos e práticas adequadas para sistematicamente elicitar, documentar, validar e gerenciar os requisitos. Mesmo quando um sistema é desenvolvido de forma ad hoc, uma abordagem sistemática e disciplinada da ER (por exemplo, promovendo sistematicamente o entendimento compartilhado, ver Princípio 3) irá melhorar a qualidade do sistema resultante.

> A agilidade e a flexibilidade não são desculpas válidas para um estilo de trabalho desordenado e ad hoc na ER.

Entretanto, não há um processo universal de ER nem um conjunto universal de práticas de ER que funcionam bem em cada situação ou pelo menos na maioria das situações: não há um "tamanho único" na ER.

Trabalho sistemático e disciplinado significa que os Engenheiros de Requisitos:

- Configuram um processo de ER bem adaptado ao problema em questão e bem ajustado ao processo usado para desenvolver o sistema (ver Capítulo 5).
- Selecionam, a partir do conjunto de práticas e produtos de trabalho da ER disponíveis, aqueles mais adequados ao problema, contexto e ambiente de trabalho em questão (ver Capítulos 3, 4 e 6).
- Não usam sempre o mesmo processo e produtos de trabalho.
- Não reutilizam processos e práticas da ER empregados com sucesso em projetos anteriores.

2.3 Leitura adicional

Glinz [Glin2008] discute o valor dos requisitos de qualidade e dos requisitos em geral [Glin2016].

Glinz e Wieringa [GlWi2007] explicam a noção e a importância dos stakeholders.

Glinz e Fricker [GlFr2015] discutem o papel e a importância do entendimento compartilhado.

As publicações de Jackson [Jack1995b] e Gunter et al. [GGJZ2000] são fundamentais para o problema dos requisitos no contexto. O papel do contexto na ER também é discutido por Pohl [Pohl2010].

Gause e Weinberg [GaWe1989] discutem a interdependência de problemas e soluções. Swartout e Balzer [SwBa1982] foram os primeiros a salientar que criar uma especificação completa antes de iniciar a implementação raramente é possível.

A validação é coberta em qualquer livro didático sobre ER. Grünbacher e Seyff [GrSe2005] discutem como chegar a um acordo através da negociação de requisitos.

Kano et al. [KSTT1984] estavam entre os primeiros a enfatizar o papel da inovação. Maalej, Nayebi, Johann, e Ruhe [MNJR2016] discutem o uso de feedback explícito e implícito do desenvolvedor na ER. Maiden, Gitzikis, e Robertson [MaGR2004] discutem como a criatividade pode fomentar a inovação na ER. Gorschek et al. [GFPK2010] delineia um processo sistemático de inovação.

3 Produtos de Trabalho e Práticas de Documentação

A Engenharia de Requisitos (ER) tradicional exige a redação de uma especificação de requisitos abrangente, completa e sem ambiguidades [IEEE830], [Glin2016]. Embora ainda seja apropriado criar especificações de requisitos completas em muitos casos, há também muitos outros casos em que o custo de escrever tais especificações excede seu benefício. Por exemplo, especificações de requisitos completas são úteis ou até mesmo necessárias quando se trata de licitação ou terceirização do projeto e implementação de um sistema ou quando um sistema é crítico para a segurança e a conformidade regulamentar é necessária. Por outro lado, quando os stakeholders e os desenvolvedores unem forças para definir e desenvolver um sistema iterativamente, não faz sentido escrever uma especificação abrangente dos requisitos. Portanto, é vital na ER adaptar a documentação ao contexto do projeto e selecionar produtos de trabalho para documentar os requisitos e suas as informações relacionadas que produzam um valor otimizado para o projeto.

Neste capítulo, você aprenderá sobre os produtos de trabalho típicos da ER e como criá-los.

3.1 Produtos de Trabalho na Engenharia de Requisitos

Há uma variedade de produtos de trabalho que são utilizados na ER.

> **Definição 3.1. Produto de Trabalho:**
> O registro de resultados intermediários ou finais gerados por um processo de trabalho.

Consideramos o termo *artefato* como um sinônimo de produto de trabalho. Preferimos o termo produto de trabalho em vez de artefato para expressar a conotação de que um produto de trabalho é o resultado de um trabalho realizado em um processo de trabalho.

De acordo com esta definição, um produto de trabalho da ER pode ser qualquer coisa que expresse requisitos, desde uma única frase ou diagrama até uma especificação de requisitos de sistema que cubra centenas de páginas. Note que um produto de trabalho pode conter outros produtos de trabalho.

3.1.1 Características dos Produtos de Trabalho

Os produtos de trabalho podem ser caracterizados por sua: *finalidade*, *tamanho*, *representação*, *vida útil* e *armazenamento*.

Tabela 3.1 dá uma visão geral dos produtos de trabalho típicos utilizados na ER juntamente com sua respectiva finalidade (ou seja, o que o produto de trabalho especifica ou fornece) e

tamanho típico. A tabela está estruturada em quatro grupos: produtos de trabalho para requisitos individuais, conjuntos de requisitos, estruturas de documentação e outros produtos de trabalho.

Há muitas maneiras diferentes de representar um produto de trabalho. Na ER, representações baseadas em *linguagem natural*, *templates* e *modelos* são de particular importância. Estes são discutidos nas seções 3.2, 3.3, e 3.4, respectivamente. Há outras representações, tais como desenhos ou protótipos, que são abordadas na seção 3.7.

Todo produto de trabalho tem uma *vida útil*. Este é o período de tempo desde a criação do produto de trabalho até o ponto em que o produto de trabalho é descartado ou se torna irrelevante. Distinguimos três categorias de produtos de trabalho com respeito à vida útil: produtos de trabalho *temporários*, *evolutivos* e *duráveis*.

Produtos de trabalho temporários são criados para apoiar a comunicação e criar um entendimento compartilhado (por exemplo, um esboço de uma interação usuário-sistema criado em um workshop). Produtos de trabalho temporários são descartados após o uso; nenhum metadado é mantido sobre esses produtos de trabalho.

Os *produtos de trabalho evolutivos* emergem durante várias iterações ao longo do tempo (por exemplo, uma coleção de Histórias de Usuários que cresce tanto no número de histórias quanto no conteúdo da história). Alguns metadados (pelo menos o autor, status e histórico de revisão) devem ser mantidos para cada produto de trabalho em evolução. Dependendo da importância e do status de um produto de trabalho, os procedimentos de controle de mudança precisam ser aplicados quando se modifica um produto de trabalho em evolução.

Produtos de trabalho duráveis foram oficialmente liberados ou configurados como baseline (p. ex., uma especificação de requisitos que faz parte de um contrato ou um backlog de sprint que é implementado em uma determinada iteração). Um conjunto completo de metadados deve ser mantido para administrar adequadamente o produto de trabalho e um elaborado processo de mudança deve ser seguido para mudar um produto de trabalho durável (Capítulo 6).

Um produto de trabalho temporário pode se tornar um produto de trabalho evolutivo quando os Engenheiros de Requisitos decidem manter um produto de trabalho e desenvolvê-lo ainda mais. Neste caso, alguns metadados devem ser adicionados a fim de manter a evolução do produto de trabalho sob controle. Quando um produto de trabalho evolutivo é oficialmente liberado ou configurado como uma baseline, ele muda seu status de vida útil evolutivo para durável.

Tabela 3.1 Visão geral dos produtos de trabalho da ER

Produto de trabalho	Objetivo: O produto de trabalho especifica/fornece	Tamanho*
Requisitos únicos		
Requisito individual	Um único requisito, tipicamente em forma de texto	P
História de Usuário	Uma função ou comportamento do ponto de vista de um stakeholder	P
Conjuntos coerentes de requisitos		
Caso de uso	Uma função do sistema a partir da perspectiva de um ator ou usuário	P–M
Modelo gráfico	Vários aspectos, por exemplo, contexto, função, comportamento (ver seção 3.4)	M
Descrição de tarefa	Uma tarefa que um sistema deve realizar	P–M
Descrição de interface	As informações trocadas entre um sistema e um ator no contexto do sistema	M
Épico	Uma visão de alto nível sobre a necessidade de um stakeholder	M
Funcionalidade	Uma funcionalidade, capacidade ou característica marcante de um sistema	P–M
Documentos Estruturados		
Especificação de requisitos de sistema**	Um documento abrangente de requisitos	G–XG
Backlog de Produto e de Sprint	Uma lista de itens de trabalho, incluindo requisitos	M–G
Mapa de histórias	Um arranjo visual das histórias dos usuários	M
Visão	Uma concepção imaginária de um sistema futuro	M
Outros produtos de trabalho		
Glossário	Terminologia comum unívoca e acordada	M

Produto de trabalho	Objetivo: O produto de trabalho especifica/fornece	Tamanho*
Nota textual ou esboço gráfico	Um memorando para a comunicação e compreensão	P
Protótipo	Uma especificação com exemplos, particularmente para entender, validar e negociar requisitos	P–G

*: P: Pequeno, M: Médio, G: Grande, XG: Extra grande
**: Outros exemplos são: especificação de requisitos de negócios, especificação de requisitos de domínio, especificação de requisitos dos stakeholders/usuários ou especificação de requisitos de software

Hoje a maioria dos produtos de trabalho são armazenados digitalmente em arquivos, bancos de dados ou ferramentas de ER. Produtos de trabalho informais e temporários também podem ser armazenados em outros meios, por exemplo, anotações ou cartões em um quadro Kanban.

3.1.2 Níveis de Abstração

Os requisitos e seus produtos de trabalho correspondentes ocorrem em vários níveis de abstração – desde, por exemplo, requisitos de alto nível para um novo processo negocial, até requisitos em um nível muito detalhado, como a reação de um componente de software específico a um evento excepcional.

Os requisitos de negócio, de domínio, de stakeholder e de usuário normalmente ocorrem em um nível de abstração mais elevado do que os requisitos de sistema e de software. Quando um sistema consiste em uma hierarquia de subsistemas e componentes, temos requisitos de sistema nos níveis de abstração correspondentes para subsistemas e componentes.

Quando os requisitos de negócio e os requisitos de stakeholder são expressos em produtos de trabalho concretos (tais como especificações de requisitos de negócio, especificações de requisitos de stakeholders ou documentos de visão) eles precedem a especificação de requisitos de sistema. Por exemplo, em situações contratuais, onde um cliente solicita o desenvolvimento de um sistema a um fornecedor, o cliente frequentemente cria e libera uma especificação de requisitos de stakeholder. O fornecedor então usa-o como base para produzir uma especificação de requisitos do sistema. Em outros projetos, os requisitos de negócio, de stakeholder e de sistema podem evoluir conjuntamente.

Alguns produtos de trabalho, como requisitos individuais, esboços ou modelos de processo, ocorrem em todos os níveis. Outros produtos de trabalho estão especificamente associados a certos níveis. Por exemplo, uma especificação dos requisitos do sistema está associada ao nível do sistema. Observe que um requisito individual em alto nível de abstração pode ser refinado em vários requisitos detalhados em níveis mais concretos.

A escolha do nível adequado de abstração depende particularmente do tema a ser especificado e da finalidade da especificação. Por exemplo, se o assunto a ser especificado for uma parte de baixo nível do problema a ser resolvido, ele será especificado em um nível de abstração bastante baixo. É importante, contudo, não misturar requisitos que estão em diferentes níveis de abstração. Por exemplo, na especificação de um sistema de informação de saúde, ao escrever um requisito detalhado sobre fotos nos cartões de identificação de clientes, o parágrafo subsequente não deve declarar um objetivo geral do sistema, como a redução do custo da saúde, mantendo o nível atual de serviço para os clientes. Em produtos de trabalho de pequeno e médio porte (por exemplo, Histórias de Usuário ou Casos de Uso), os requisitos devem estar aproximadamente no mesmo nível de abstração. Em grandes produtos de trabalho, como uma especificação de requisitos de sistema, os requisitos em diferentes níveis de abstração devem ser separados, estruturando a especificação de forma adequada (seção 3.6).

Os requisitos ocorrem naturalmente em diferentes níveis de abstração. É útil selecionar produtos de trabalho adequados para um determinado nível de abstração e estruturar adequadamente produtos de trabalho que contenham requisitos em múltiplos níveis de abstração.

3.1.3 Nível de Detalhe

Ao especificar os requisitos, os Engenheiros de Requisitos têm que decidir sobre o nível de detalhe em que os requisitos devem ser especificados. Entretanto, decidir qual nível de detalhe é apropriado ou mesmo ótimo para um determinado requisito é uma tarefa desafiadora.

Por exemplo, em uma situação em que o cliente e o fornecedor de um sistema colaboram estreitamente, pode ser suficiente declarar um requisito sobre um formulário de entrada de dados da seguinte forma: "O sistema deve fornecer um formulário para a entrada dos dados pessoais do cliente". Em contraste, em uma situação onde o projeto e a implementação do sistema são terceirizados para um fornecedor com pouco ou nenhum conhecimento de domínio, será necessária uma especificação detalhada do formulário de entrada do cliente.

O nível de detalhe em que os requisitos serão especificados depende de vários fatores, em particular:

- O problema e o contexto do projeto: quanto mais difícil o problema e menos familiarizados os engenheiros e desenvolvedores de requisitos estão com o contexto do projeto, mais detalhes são necessários.
- O grau de entendimento compartilhado do problema: quando há pouco entendimento compartilhado implícito (ver Princípio 3 no Capítulo 2), são necessárias especificações explícitas e detalhadas para criar o grau necessário de entendimento compartilhado.
- O grau de liberdade deixado aos projetistas e programadores: requisitos menos detalhados dão mais liberdade aos desenvolvedores.

- Disponibilidade de feedback rápido dos stakeholders durante o projeto e a implementação: quando o feedback rápido está disponível, especificações menos detalhadas são suficientes para controlar o risco de desenvolvimento do sistema errado.
- Custo vs. valor de uma especificação detalhada: quanto maior o benefício de um requisito, mais podemos nos dar ao luxo de especificá-la em detalhes.
- Normas e regulamentos: Normas impostas e restrições regulatórias podem significar que os requisitos têm que ser especificados com mais detalhes do que seria necessário de outra forma.

Não existe um nível de detalhamento universalmente "certo" para os requisitos. Para cada requisito, o nível adequado de detalhes depende de muitos fatores. Quanto maior o nível de detalhe na especificação dos requisitos, menor será o risco de eventualmente se obter algo que tenha características ou propriedades inesperadas ou ausentes. No entanto, o custo da especificação aumenta à medida que o nível de detalhe aumenta.

3.1.4 Aspectos a serem considerados

Independentemente dos produtos de trabalho de ER utilizados, vários aspectos devem ser considerados ao especificar os requisitos [Glin2019].

Primeiro, como existem requisitos funcionais, requisitos de qualidade e restrições (ver Seção 1.1), os Engenheiros de Requisitos têm que se certificar de que eles cobrem todos os três tipos de requisitos ao documentar os requisitos. Na prática, os stakeholders tendem a omitir os requisitos de qualidade porque as consideram como assegurados.

Eles também tendem a especificar restrições como requisitos funcionais. É importante, portanto, que os Engenheiros de Requisitos incluam os três tipos de requisitos.

Ao observarmos os requisitos funcionais, observamos que elas se referem a diferentes perspectivas, como, por exemplo, uma estrutura de dados necessária, uma ordem de ações necessária ou a reação necessária a algum evento externo. Fazemos distinção entre três perspectivas principais: *estrutura e dados*, *função e fluxo*, e *estado e comportamento*.

O aspecto de *estrutura* e *dados* focam nos requisitos relativos à estrutura estática do sistema e nos dados (persistentes) que ele deve conhecer para executar as funções requeridas entregando os resultados requeridos.

O aspecto de *função* e *fluxo* lida com as funções que um sistema deve fornecer e o fluxo de controle e dados dentro e entre as funções para criar os resultados necessários a partir de dados de entrada.

O aspecto de *estado e comportamento* concentra-se em especificar o comportamento dependente do estado de um sistema – em particular, como um sistema deve reagir a qual evento externo, dependendo do estado atual do sistema.

Ao tratar de *requisitos de qualidade*, tais como usabilidade, confiabilidade ou disponibilidade, um modelo de qualidade – por exemplo, o modelo fornecido pela ISO/IEC 25010 [ISO25010]– pode ser usado como um checklist.

Dentro dos requisitos de qualidade, os *requisitos de desempenho* são de particular importância. Os requisitos de desempenho tratam de:

- Tempo (por exemplo, para executar uma tarefa ou reagir a eventos externos)
- Volume (por exemplo, tamanho do banco de dados necessário)
- Frequência (por exemplo, de computar uma função ou receber estímulos dos sensores)
- Taxa de transferência (por exemplo, taxas de transmissão de dados ou transações)
- Consumo de recursos (por exemplo, CPU, armazenamento, largura de banda, bateria)

Algumas pessoas também consideram a precisão necessária de um cálculo como um requisito de desempenho.

Sempre que possível, devem ser especificados valores mensuráveis. Quando os valores seguem uma distribuição de probabilidade, especificar apenas a média não é suficiente. Se a função de distribuição e seus parâmetros não puderem ser especificados, os Engenheiros de Requisitos devem se esforçar para especificar valores mínimos e máximos ou valores de 95% além das médias.

Documentar requisitos de qualidade além dos requisitos de desempenho é notoriamente difícil.

Representações qualitativas, tais como "O sistema deve ser seguro e fácil de usar", são ambíguas e, portanto, difíceis de serem alcançadas e validadas.

As *representações quantitativas* são mensuráveis, o que é um grande trunfo em termos de alcançar e validar sistematicamente um requisito de qualidade. Entretanto, elas levantam, por definição, dificuldades (por exemplo, como podemos especificar a segurança em termos quantitativos?) e podem ser muito caros para especificar.

As representações operacionalizadas estabelecem requisitos de qualidade em termos de requisitos funcionais para alcançar a qualidade desejada. Por exemplo, um requisito de segurança de dados pode ser expresso em termos de uma função de login que restringe o acesso aos dados e uma função que encripta os dados armazenados. As representações operacionalizadas tornam os requisitos de qualidade testáveis, mas também podem implicar em decisões prematuras de projeto.

A regra frequentemente ouvida "Somente um requisito de qualidade quantificado é um bom requisito de qualidade" está ultrapassada e pode levar a requisitos de qualidade com valor baixo ou mesmo negativo devido ao alto esforço envolvido na quantificação. Em vez disso, uma abordagem baseada no risco deve ser utilizada [Glin2008].

As *representações qualitativas* dos requisitos de qualidade *são suficientes* nas seguintes situações:

- Há entendimento compartilhado implícito suficiente entre os stakeholders, os Engenheiros de Requisitos e os desenvolvedores.
- stakeholders, Engenheiros de Requisitos e desenvolvedores concordam que uma solução conhecida satisfaz os requisitos.
- stakeholders só querem dar orientações gerais de qualidade e confiar nos desenvolvedores para obter os detalhes corretos.
- Ciclos de feedback curtos estão em vigor para que os problemas possam ser detectados precocemente.

Quando os desenvolvedores são *capazes de generalizar a partir de exemplos*, especificar os requisitos de qualidade em termos de exemplos quantificados ou comparações com um sistema existente temos uma forma barata e eficaz de documentar os requisitos de qualidade.

Somente nos casos em que existe um *alto risco* de não atendimento das necessidades dos stakeholders, particularmente quando os requisitos de qualidade forem *críticos para a segurança*, deverá ser considerada uma representação quantificada ou uma representação operacionalizada em termos de requisitos funcionais.

Ao especificar *restrições*, as seguintes categorias de restrições devem ser consideradas:

- *Técnica*: determinadas interfaces ou protocolos, componentes ou estruturas que têm de ser utilizadas etc.
- *Legal*: restrições impostas por leis, contratos, normas ou regulamentos
- *Organizacional*: pode haver restrições em termos de estruturas organizacionais, processos ou políticas que não devem ser alteradas pelo sistema.
- *Cultural*: os hábitos e expectativas dos usuários são, em certa medida, moldados pela cultura em que vivem. Este é um aspecto particularmente importante a considerar quando os usuários de um sistema vêm de culturas diferentes ou quando os engenheiros de requisitos e desenvolvedores estão enraizados em uma cultura diferente da dos usuários do sistema.
- *Ambiental*: ao especificar sistemas ciberfísicos, condições ambientais tais como temperatura, umidade, radiação ou vibração podem ter que ser consideradas como restrições; consumo de energia e dissipação de calor podem constituir outras restrições.
- *Física*: quando um sistema incorpora componentes físicos ou com eles interage, o sistema é limitado pelas leis da física e pelas propriedades dos materiais utilizados para os componentes físicos.
- Além disso, *soluções específicas ou restrições particulares exigidas por importantes stakeholders* também constituem restrições.

Finalmente, os requisitos só podem ser entendidos no *contexto* (ver Princípio 4 no Capítulo 2). Consequentemente, um aspecto adicional tem que ser considerado, que chamamos de *contexto e limite*.

O *contexto* e *limite* abrange os requisitos de domínio e as suposições de domínio no contexto do sistema, bem como os actores externos que o sistema interage e as interfaces externas entre o sistema e seu ambiente no seu limite.

Há muitas relações e dependências entre os aspectos mencionados acima. Por exemplo, uma solicitação emitida por um usuário (contexto) pode ser recebida pelo sistema através de uma interface externa (limite), acionar uma transição de estado do sistema (estado e comportamento), que inicia uma ação (função) seguida por outra ação (fluxo) que requer dados com alguma estrutura (estrutura e dados) para fornecer um resultado ao usuário (contexto) dentro de um determinado intervalo de tempo (qualidade).

Alguns produtos de trabalho focam em um aspecto específico e abstraem os outros aspectos. Isto é particularmente verdadeiro ao modelar requisitos (Seção 3.4). Outros produtos de trabalho, como uma especificação dos requisitos do sistema, cobrem todos os aspectos citados. Quando aspectos diferentes são documentados em produtos de trabalho separados ou em capítulos separados do mesmo produto de trabalho, esses produtos de trabalho ou capítulos de trabalho devem ser mantidos consistentes entre si.

Muitos aspectos diferentes precisam ser considerados ao documentar os requisitos, em particular, funcionalidade (estrutura e dados, função e fluxo, estado e comportamento), qualidade, restrições e contexto circundante (contexto e limite).

3.1.5 Diretrizes Gerais de Documentação

Independentemente das técnicas utilizadas, há algumas diretrizes gerais que devem ser seguidas na criação de produtos de trabalho na ER:

- Selecionar um tipo de produto de trabalho que se adapte à finalidade pretendida.
- Evitar redundância referenciando o conteúdo em vez de repetir o mesmo conteúdo novamente.
- Evite inconsistências entre produtos de trabalho, particularmente quando cobrem diferentes aspectos.
- Usar termos de forma consistente, conforme definido no glossário.
- Estruturar os produtos de trabalho adequadamente, por exemplo, usando estruturas padronizadas e/ou templates.

3.1.6 Planejamento de Produtos de Trabalho

Cada cenário de projeto e cada domínio são diferentes, portanto, o conjunto de produtos de trabalho resultante deve ser definido para cada empreendimento. As partes envolvidas, particularmente os Engenheiros de Requisitos, stakeholders, proprietários ou gerentes de projetos/produtos precisam concordar sobre as seguintes questões:

- Em quais produtos de trabalho os requisitos devem ser registados e para quais finalidades (veja tabela Tabela 3.1)?
- Quais níveis de abstração devem ser considerados (Seção 3.1.2)?

- Qual nível de detalhamento dos requisitos em cada nível de abstração (Seção 3.1.3)?
- Como os requisitos devem ser representados nesses produtos de trabalho (por exemplo, baseados na linguagem natural ou em modelos, ver a seguir) e quais notações devem ser utilizadas?

Os Engenheiros de Requisitos devem definir os produtos de trabalho da ER a serem utilizados em uma fase inicial de um projeto. Esta definição antecedente:

- Ajuda no planejamento de esforços e recursos
- Assegurar que notações apropriadas são utilizadas
- Assegura que todos os resultados são registrados nos produtos de trabalho corretos
- Assegura que não será necessária uma grande reorganização de informações e esforço de edição final
- Ajuda a evitar a redundância, resultando em menos trabalho e facilidade na manutenção

3.2 Produtos de Trabalho baseados em Linguagem Natural

A linguagem natural, tanto na forma falada quanto na escrita, sempre foi um meio essencial para a comunicação dos requisitos dos sistemas. O uso de linguagem natural para escrever produtos de trabalho na ER tem muitas vantagens. Em particular, a linguagem natural é extremamente expressiva e flexível, o que significa que quase qualquer requisito concebível em qualquer aspecto pode ser expresso em linguagem natural. A linguagem natural é usada na vida cotidiana e ensinada na escola, não é necessário nenhum treinamento específico para ler e compreender textos em linguagem natural.

A evolução humana moldou a linguagem natural como um meio de *comunicação oral entre pessoas que interagem diretamente*, onde mal-entendidos e informações ausentes podem ser detectados e corrigidos rapidamente. Portanto, a linguagem natural *não é* otimizada para uma comunicação precisa, inequívoca e abrangente por meio de documentos escritos. Isto constitui um grande problema ao escrever a documentação técnica (como requisitos) em linguagem natural. Ao contrário da comunicação em linguagem natural *falada*, onde a comunicação é contextualizada e interativa com feedback imediato, não há meios naturais para detectar e corrigir rapidamente ambiguidades, omissões e inconsistências em textos *escritos* em linguagem natural. Pelo contrário, encontrar tais ambiguidades, omissões e inconsistências em textos escritos é difícil e caro, particularmente para produtos de trabalho que contêm uma grande quantidade de textos em linguagem natural.

O problema pode ser mitigado até certo ponto escrevendo conscientemente a documentação técnica, seguindo regras comprovadas e evitando armadilhas conhecidas.

Ao escrever os requisitos em linguagem natural, os Engenheiros de Requisitos podem evitar muitos mal-entendidos potenciais aplicando algumas regras simples:

- Escreva frases curtas e bem estruturadas. A regra geral é expressar um único requisito em uma única frase em linguagem natural. Para conseguir uma boa

estrutura, os Engenheiros de Requisitos devem usar templates de frases (Seção 3.33.3).

- Crie produtos de trabalho bem estruturados. Além de escrever frases bem estruturadas (ver acima), os produtos de trabalho escritos em linguagem natural também devem ser bem estruturados como um todo. Uma maneira comprovada de fazer isso é utilizar uma estrutura hierárquica de partes, capítulos, seções e subseções, como geralmente é feito em livros técnicos. Templates de documentos (Seção 3.3) ajudam você a obter uma boa estrutura.
- Defina e use uma terminologia uniforme e consistente. Criar e utilizar um glossário (Seção 3.5) é o meio principal para evitar mal-entendidos e inconsistências sobre a terminologia.
- Evite o uso de termos e frases vagas, imprecisas ou ambíguas.
- Conheça e evite as armadilhas da escrita técnica (ver abaixo).

Ao escrever documentos técnicos em linguagem natural, há algumas armadilhas bem conhecidas que devem ser evitadas ou usadas com cuidado (veja por exemplo [GoRu2003]).

Engenheiros de Requisitos devem *evitar* escrever requisitos que contenham:

- *Descrições incompletas.* Os verbos em linguagem natural normalmente vêm com um conjunto de marcadores para substantivos ou pronomes. Por exemplo, o verbo "dar" tem três marcadores: *quem* dá *o quê* a *quem*. Ao escrever um requisito em linguagem natural, todos os marcadores do verbo utilizado devem ser preenchidos.
- *Substantivos inespecíficos.* O uso de substantivos como "os dados" ou "o usuário" deixa muito espaço para diferentes interpretações por parte de diferentes stakeholders ou desenvolvedores. Eles devem ser substituídos por substantivos mais específicos ou tornados mais específicos, acrescentando adjetivos ou atribuindo-lhes um tipo bem definido.
- *Condições incompletas.* Ao descrever o que deve ser feito, muitas pessoas se concentram no caso normal, omitindo casos alternativos e excepcionais. Na escrita técnica, isto é uma armadilha a ser evitada: quando algo acontece somente se certas condições forem verdadeiras, tais condições devem ser declaradas, especificando tanto cláusulas *então* como *senão*.
- *Comparações incompletas.* Na comunicação oral, as pessoas tendem a usar comparativos (por exemplo, "o novo aplicativo de vídeo é muito melhor") sem dizer com o que comparando, tipicamente assumindo que isto está claro a partir do contexto. Na escrita técnica, as comparações devem incluir um objeto de referência, por exemplo, "mais rápido *que 0,1 ms*".

Há mais algumas coisas que os Engenheiros de Requisitos precisam usar com cuidado, pois elas constituem potenciais armadilhas:

- *Voz passiva.* Frases em voz passiva não tem sujeito explícito. Se um requisito for declarado na voz passiva, isto pode esconder quem é responsável pela ação descrita no requisito, levando a uma descrição incompleta.

- *Quantificadores universais*. Quantificadores universais são palavras como *todas*, *sempre*, ou *nunca*, que são usadas para fazer afirmações que são universalmente verdadeiras. Em sistemas técnicos, porém, tais propriedades universais são raras. Sempre que os Engenheiros de Requisitos usam um quantificador universal, eles precisam refletir se estão declarando uma propriedade verdadeiramente universal ou se estão especificando uma regra geral que tem exceções (que eles também precisam especificar). Eles devem aplicar a mesma cautela ao usar cláusulas "ou isso ou aquilo", que, por sua semântica, excluem quaisquer outros casos excepcionais.
- *Nominalizações*. Quando um substantivo é derivado de um verbo (por exemplo, "autenticação" de "para autenticar"), os linguistas chamam isto de uma nominalização. Ao especificar os requisitos, os Engenheiros de Requisitos precisam tratar as nominalizações com cuidado, pois uma nominalização pode ocultar requisitos não especificados. Por exemplo, o requisito "Somente após a autenticação bem-sucedida, o sistema deverá fornecer acesso a um desenvolvedor (...)" implica que existe um procedimento para autenticação de usuários. Ao escrever tal requisito, portanto, o Engenheiro de Requisitos deve verificar se também existem requisitos sobre o procedimento para autenticar usuários legítimos.

A linguagem natural é um meio muito poderoso para escrever requisitos. Para mitigar as desvantagens inerentes ao uso da linguagem natural para documentação técnica, os Engenheiros de Requisitos devem seguir regras de redação comprovadas e evitar armadilhas bem conhecidas.

3.3 Produtos de trabalho baseados em Templates

Como mencionado na seção 3.2 acima, o uso de templates é um meio comprovado para escrever bons produtos de trabalho, bem estruturados, em linguagem natural e assim mitigar algumas das fraquezas da linguagem natural para a escrita técnica. Um template é um modelo pronto para a estrutura sintática de um produto de trabalho. Ao utilizar linguagem natural na ER, distinguimos três classes de templates: templates de frases, templates de formulários e templates de documentos.

3.3.1 Templates de Frases

Definição 3.2. Template de frase:
Um gabarito para a estrutura sintática de uma frase que expressa um requisito individual ou uma história de usuário em linguagem natural.

Um template de frase fornece uma estrutura de frase esqueleto com espaços reservados, os quais os Engenheiros de Requisitos preenchem para obter frases bem estruturadas e uniformes que expressam os requisitos.

O uso de templates de frases é uma boa prática ao escrever requisitos individuais em linguagem natural e ao escrever histórias de usuários.

3.3.1.1 Templates de frases para requisitos individuais

Vários templates de Frases para escrever requisitos individuais foram definidos, por exemplo, em [ISO29148], [MWHN2009], e [Rupp2014]. A norma ISO/IEC/IEEE 29148 [ISO29148] fornece um modelo único e uniforme para os requisitos individuais, como segue:

[<Condição>] <Sujeito> <Ação> <Objetos> [<Restrição>].

Exemplo: Quando um cartão válido for detectado, o sistema exibirá a mensagem "Insira seu PIN" na tela de diálogo dentro de 200 ms.

Ao formular uma ação com este template, as seguintes convenções sobre o uso de verbos auxiliares são frequentemente utilizadas na prática:

- *Deverá* denota um requisito obrigatório.
- *Deveria* denota um requisito que não é obrigatório, mas fortemente desejado.
- *Poderia* denota um requisito não obrigatório, apenas sugerido.

Irá (ou utilizando um *verbo no tempo presente* sem um dos verbos auxiliares mencionados acima) denota uma declaração factual que não é considerada como um requisito.

Quando não há significados acordados para verbos auxiliares em um projeto, ou quando em dúvida, definições como as dadas acima devem fazer parte de uma especificação de requisitos.

EARS (Easy Approach to Requirements Syntax) [MWHN2009] fornece um conjunto de templates de frases que são adaptados para diferentes situações, conforme descrito abaixo.

Requisitos onipresentes (sempre devem ser mantidos):

> O <nome do sistema> deve <resposta do sistema>.

Requisitos acionadas por eventos (desencadeadas por um evento externo):

> [QUANDO <pré-condições opcionais>] <evento acionador> o <nome do sistema> everá <resposta do sistema>.

Requisitos para comportamentos indesejados (descreve situações a serem evitadas):

> SE <pré-condições opcionais> <evento acionador> ENTÃO o <nome do sistema> everá <resposta do sistema>.

Nota: Embora o template de requisitos para comportamentos indesejados seja similar ao modelo acionado por eventos, Mavin et al. fornecem um template separado para este último, argumentando que o comportamento indesejado (principalmente devido a eventos inesperados no contexto, tais como falhas, ataques ou coisas em que ninguém pensou), é uma importante fonte de omissões na ER.

Requisitos orientados a estados (aplicam-se somente a estados):

> QUANDO <em um estado específico> o <nome do sistema> deverá <resposta do sistema>.

Requisitos opcionais (aplicável somente se este requisito estiver presente no sistema):

> ONDE <o requisito presente> o <nome do sistema> deverá <resposta do sistema>.

Na prática, frases que combinam as palavras-chave QUANDO, ENQUANTO e ONDE podem ser necessárias para expressar requisitos mais complexos.

O EARS foi desenvolvido principalmente para a especificação de sistemas ciberfísicos. No entanto, também pode ser adaptado para outros tipos de sistemas.

3.3.1.2 Templates de Frase para Histórias de Usuários

O modelo clássico de template para escrever histórias de usuário foi introduzido por Cohn [Cohn2004]:

> Como <papel> Eu quero <requisito> Para <benefício>.

Exemplo: "Como gestor operacional, quero fazer consultas ad hoc ao sistema de contabilidade para que eu possa fazer planejamento financeiro para meu departamento".

Embora Cohn tenha definido a parte <benefício> do template como opcional, é prática padrão hoje em dia especificar um benefício para cada história de usuário.

Cada história de usuário deve ser acompanhada por um conjunto de *critérios de aceite* – isto é, critérios que a implementação da história de usuário deve satisfazer para que seja aceita pelos stakeholders. Os critérios de aceite tornam uma história de usuário mais concreta e menos ambígua. Isto ajuda a evitar erros de implementação devido a mal-entendidos.

3.3.2 Templates de Formulários

Definição 3.3. Template de formulário:
Um template de um formulário com campos pré-definidos a serem preenchidos.

Os templates de formulários são usados para estruturar produtos de trabalho de tamanho médio, tais como casos de uso. Cockburn [Cock2001] introduziu um popular template de formulário para casos de uso. [Laue2002] propôs um template para descrição de tarefas.

mostra um template de formulário simples para casos de uso. Cada etapa do fluxo pode ser subdividida em uma ação de um ator e a resposta pelo sistema.

Tabela 3.2 Um template simples de formulário para escrever casos de uso

Nome	< Uma breve frase verbal do que o caso de uso faz>
Pré-condição	<Condições que devem ser verdadeiras quando o caso de uso for disparado>
Pós Condição em caso de sucesso	<Estado final do caso de uso em caso de sucesso>
Pós-Condição em caso de falha	<Estado final do caso de uso em caso de falha>
Ator principal	<Nome do ator>
Outros atores	<Lista de outros atores envolvidos, se houver>
Evento disparador	<Evento que inicia a execução do caso de uso>
Fluxo normal	<Descrição do principal cenário de sucesso em uma sequência de etapas: <passo 1> <ação 1> <passo 2> <ação 2> ... <passo n> <ação n> ... >
Fluxos alternativos e de exceção	<Descrição de passos alternativos ou excepcionais, com referências aos passos correspondentes do fluxo normal>
Extensões	<Possíveis casos de uso que estendem este caso de uso, com referências aos passos estendidos nos fluxos normal, alternativo e de exceção>
Informações relacionadas	<Campo opcional para maiores informações, tais como desempenho, frequência, relação com outros casos de uso etc.>

Os templates de formulários também são úteis para escrever requisitos de qualidade em uma forma mensurável [Gilb1988].

Tabela 3.3 fornece um template de formulário simples para requisitos de qualidade mensuráveis, juntamente com um exemplo.

Tabela 3.3 Um template de formulário para especificar requisitos de qualidade mensuráveis

Template		Exemplo
Identificação	<Identificação do requisito>	R137.2
Objetivo	<Objetivo qualitativamente declarado>	Confirmar reservas de hotel imediatamente
Escala	<Escala para medir o requisito>	Tempo decorrido em segundos (escala de proporção)
Métrica	<Procedimento para medir o requisito>	Medir o tempo entre pressionar "Reservar" e o aplicativo exibir a confirmação da reserva. Medindo a diferença de tempo.
Mínimo	<Qualidade mínima aceitável a ser alcançada>	Menos de 5 s para pelo menos 95% de todos os casos
Intervalo OK	<Faixa de valores esperada como OK>	Entre 0,5 e 3 s para mais de 98% de todos os casos
Ótimo	<Qualidade alcançada no melhor caso possível>	Menos de 0,5 s para 100% de todos os usuários

3.3.3 Templates de Documentos

Definição 3.4. Template de documento:
Template fornecendo uma estrutura esqueleto pré-definida para um documento.

Os templates de documentos ajudam a estruturar sistematicamente os documentos de requisitos – por exemplo, uma especificação dos requisitos do sistema. Templates de documentos para a ER podem ser encontrados em normas, por exemplo, em [ISO29148]. O template Volere de Robertson e Robertson [RoRo2012], [Vole2020] também é popular na prática. Quando uma especificação de requisitos é incluída no conjunto de produtos de trabalho que um cliente encomendou e pelo qual pagou, esse cliente pode prescrever o uso de templates de documentos por ele fornecidos. Em Figura 3.1, mostramos um exemplo de um template de documento simples para uma especificação de requisitos de sistema.

3.3.4 Vantagens e Desvantagens

O uso de templates ao escrever produtos de trabalho da ER em linguagem natural tem grandes vantagens. Os templates fornecem uma estrutura clara e reutilizável para os produtos de trabalho, os fazem parecer uniformes, e assim melhoram a legibilidade dos produtos de trabalho. Os templates também o ajudam a capturar as informações mais relevantes e a cometer menos erros de omissão. Por outro lado, há uma potencial armadilha quando os Engenheiros de Requisitos usam os templates mecanicamente, concentrando-se na estrutura sintática e não no conteúdo, negligenciando tudo o que não se encaixa no template.

Parte
Parte I: Introdução
Objetivo do sistema
Escopo do Desenvolvimento do Sistema
Stakeholders
Parte II: Visão geral do sistema
Visão e Objetivos do Sistema
Contexto e limite do sistema
Estrutura geral do sistema
Características do Usuário
Parte III: Requisitos do sistema
Organizado hierarquicamente de acordo com a estrutura do sistema, usando um esquema de numeração para os requisitos
Por subsistema/componente:
Requisitos funcionais (estrutura e dados, função e fluxo, estado e comportamento)
Requisitos de qualidade
Restrições
Interfaces
Referências
Glossário (se não for gerido como um produto de trabalho próprio)
Anexos
Pressupostos e dependências

Figura 3.1 Um template simples para especificação dos requisitos do sistema

O uso de templates ao escrever produtos de trabalho na ER em linguagem natural melhora a qualidade dos produtos de trabalho, desde que os templates não sejam mal utilizados como um mero exercício sintático (preenchidos sem reflexão).

3.4 Produtos de Trabalho Baseados em Modelos

Os requisitos formulados em linguagem natural podem ser facilmente lidos pelas pessoas desde que elas saibam falar o idioma. A linguagem natural sofre de ambiguidade devido à imprecisão da semântica de palavras, frases e sentenças [Davi1993]. Esta imprecisão pode levar a confusão e omissões nos requisitos. Ao ler os requisitos textuais, você tentará interpretá-los à sua própria maneira. Muitas vezes tentamos imaginar estes requisitos em nossa mente. Quando o número de requisitos é administrável, é possível manter uma visão geral dos requisitos textuais. Quando o número de requisitos textuais se torna "muito grande", perdemos a visão geral. Esse limite é diferente para cada pessoa. O número de requisitos textuais não é a única razão para perder o discernimento e a visão geral. A complexidade dos requisitos, a relação entre os requisitos e a abstração dos requisitos também contribuem para isso. Você pode ter que ler várias vezes os requisitos formulados em linguagem natural antes de obter uma imagem correta e completa do que o sistema deve cumprir. Temos uma capacidade limitada de processar os requisitos em linguagem natural.

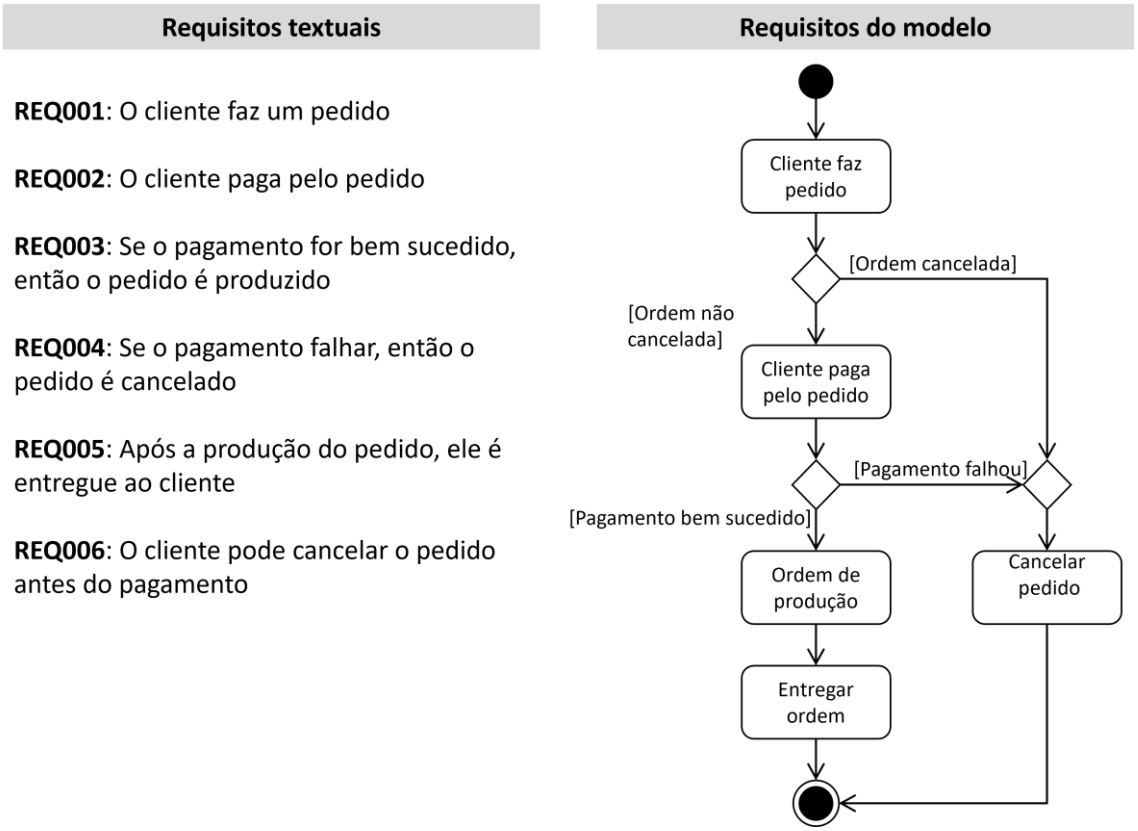


Figura 3.2 Requisitos textuais versus requisitos modelados

Um modelo é uma representação abstrata de uma parte existente da realidade ou de uma parte da realidade a ser criada. A exibição dos requisitos com um modelo (ou imagem) contribuirá para que os leitores compreendam os requisitos. Tal representação esquemática de um modelo é chamada de diagrama.

O diagrama em Figura 3.2 mostra num relance o que o sistema deve fazer, mas somente se você dominar a linguagem de modelagem. É evidente que se você não entender o diagrama, neste caso um diagrama de atividades UML, a imagem não contribuirá para um melhor entendimento dos requisitos.

Na próxima seção (3.4.1), é explicado o conceito de um modelo de requisitos. A modelagem dos requisitos e objetivos negociais é explicada na Seção 3.4.6. Um método importante para descrever os limites de um sistema é o modelo de contexto. Exemplos do contexto são descritos na Seção 3.4.2. As Seções 3.4.3 a 0 dão uma série de exemplos de linguagens de modelagem que são frequentemente usadas na prática da engenharia de sistemas.

3.4.1 O Papel dos Modelos na Engenharia de Requisitos

Como qualquer idioma, uma linguagem de modelagem consiste em regras gramaticais e uma descrição do significado das construções linguísticas, ver Seção 3.4.1.1. Embora um modelo seja uma representação visual da realidade, as regras da linguagem são importantes para entender o modelo e as nuances do modelo.

Nem sempre é eficiente ou eficaz resumir os requisitos em um modelo. Ao entender as propriedades de um modelo, podemos determinar melhor quando podemos aplicar qual modelo, ver Seção 3.4.1.2.

Assim como a linguagem natural tem vantagens e desvantagens para expressar os requisitos, o mesmo acontece com os modelos. Se observarmos estes fatos na aplicação de um modelo, podemos determinar melhor o valor agregado da aplicação do modelo "correto". Isto é discutido na Seção 3.4.1.3.

Muitos modelos já foram padronizados e são utilizados em vários campos de aplicação, ver Seção 3.4.1.4. Considere, por exemplo, a construção de uma casa, onde um arquiteto usa um modelo padronizado para descrever a casa. Exemplos para modelos fora do domínio da engenharia de software são os modelos de informação de construção (BIM) [ISO19650], que modelam os elementos necessários para planejar, construir e gerenciar edifícios e outros elementos de construção.

Outro exemplo é a eletrônica, onde o desenho dos diagramas eletrônicos é padronizado para que os profissionais possam entender, calcular e realizar a eletrônica.

Para determinar se um diagrama é aplicado corretamente, podemos validar os critérios de qualidade de um diagrama. Estes critérios são descritos na Seção 3.4.1.5.

3.4.1.1 Sintaxe e Semântica

Se você pensa em uma língua natural, por exemplo, sua língua nativa, ela é definida por sua gramática e semântica.

A gramática descreve os elementos (palavras e sentenças) e as regras que a língua deve obedecer. Em uma linguagem de modelagem, isto é chamado de sintaxe, ver Figura 3.3. A sintaxe descreve quais elementos de notação (símbolos) são usados na linguagem. Também descreve como esses elementos de notação podem ser usados em combinação.

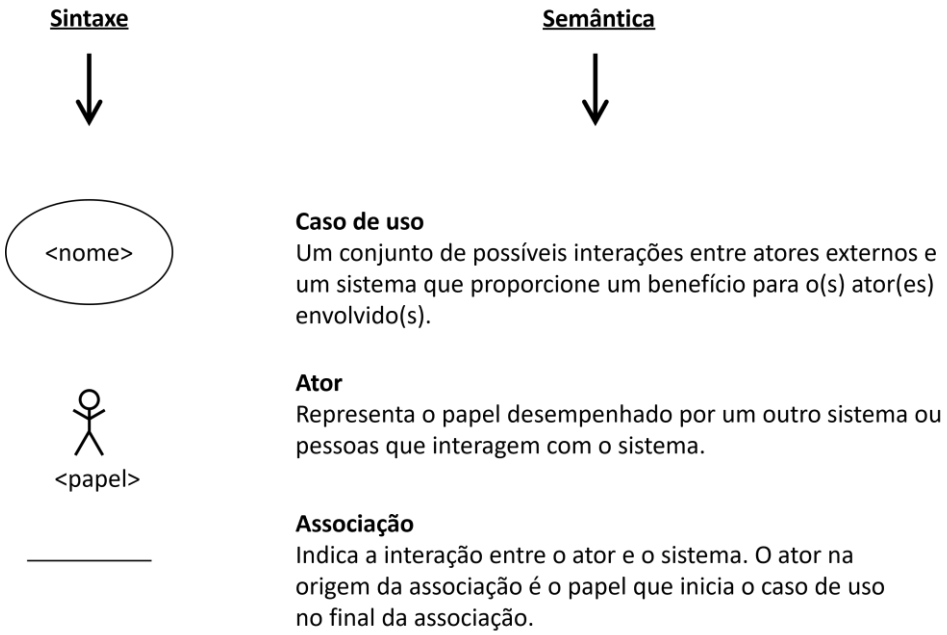


Figura 3.3 Modelagem da sintaxe e da semântica da linguagem

A semântica define o significado dos elementos da notação e define o significado da combinação de elementos. A compreensão do significado dos elementos da notação é fundamental para evitar o risco de que o modelo seja mal interpretado.

3.4.1.2 Propriedades de um modelo

Um modelo de requisitos é um modelo conceitual que retrata os requisitos para o sistema a ser desenvolvido. Um modelo também é usado para representar a situação atual para compreender, analisar e explorar os problemas atuais. Neste contexto, conceitual significa que a realidade é reduzida à sua essência. Um modelo tem um alto nível de abstração e reduz a realidade ao que é relevante neste nível genérico.

Uma linguagem de modelagem conceitual pode ser padronizada (internacionalmente) e é então referida como uma linguagem de modelagem formal. Um exemplo disso é a linguagem de modelagem UML (Unified Modeling Language), amplamente difundida e frequentemente aplicada.

Um modelo tem uma série de propriedades que são exploradas mais a fundo nas seções seguintes:

- Um modelo é feito para um propósito específico.
- Um modelo oferece uma representação da realidade.

- Um modelo é usado para reduzir a informação para que possamos melhor entender a realidade ou nos concentrar em parte dela.

Um modelo é uma representação abstrata de uma parte existente da realidade ou de uma parte da realidade a ser criada. A noção da realidade inclui qualquer conjunto concebível de elementos, fenômenos ou conceitos, incluindo outros modelos. A parte modelada da realidade é chamada de original. O processo para descrever o original pode ser descritivo ou prescritivo.

A modelagem do original existente é chamada de modelagem descritiva. Um modelo descritivo mostra a realidade atual e reflete os requisitos que são atendidos. Se ainda não há um modelo original disponível, tal modelo é o resultado da análise da situação atual.

A modelagem de um original a ser criado é chamada de modelagem prescritiva. Um modelo prescritivo indica qual é a realidade futura esperada ou necessária. Se existir um modelo descritivo para a situação em questão, então um modelo com caracteristicas prescritivas pode ser derivado do original indicando quais os requisitos que serão novos, alterados ou que não sejam mais necessários. O modelo prescritivo descreve a situação futura final desejada.

A realidade pode ser complexa. Se aplicarmos "demasiados" detalhes, um modelo pode ser difícil de entender. Esta complexa realidade pode ser simplificada pela redução da quantidade de informações no modelo. Em um modelo, podemos omitir informações irrelevantes. A redução da quantidade de informações pode nos dar uma melhor compreensão da realidade e nos permitir compreender mais facilmente a essência desta realidade. Com base na finalidade pretendida (primeira propriedade) para a qual o modelo é aplicado, apenas as informações relevantes são exibidas no modelo.

Observe que, se reduzirmos "demasiado" a informação, pode restar uma imagem nublada ou incorreta da realidade. Assim, deve-se considerar cuidadosamente a quantidade de informação que pode ser reduzida sem distorcer a realidade.

Há várias maneiras de reduzir a informação:

- Por simplificação ou agregação

  Agregar informação é uma forma de tornar a informação mais abstrata. A informação é despojada de detalhes irrelevantes e, portanto, é mais compacta. As informações são, por assim dizer, condensadas.

- Por seleção

  Ao selecionar apenas as informações relevantes, e não tudo, é possível focar no assunto em consideração. O foco está em uma parte específica ou no número de partes do total.

As duas formas de reduzir a informação também podem ser aplicadas em conjunto.

Um modelo é uma representação da realidade e cada modelo representa certos aspectos da realidade. Por exemplo, um desenho de construção mostra a divisão do espaço em um edifício e um diagrama elétrico mostra a fiação do circuito elétrico.

Os dois modelos representam o edifício para um propósito específico. Um modelo é feito para um propósito específico em um contexto específico. No exemplo acima, o contexto é o projeto e/ou a realização de um edifício. Os vários desenhos de construção representam informações sobre um aspecto específico do edifício. Isto deixa imediatamente claro que um modelo específico só pode ser usado quando se adequar ao propósito para o qual o modelo foi feito.

3.4.1.3 Vantagens e Desvantagens da Modelagem de Requisitos

Em comparação com os idiomas naturais, os modelos têm as seguintes vantagens, entre outras:

- Os elementos e suas conexões são mais fáceis de entender e de lembrar.

  Uma imagem diz mais que mil palavras. Uma imagem, e também um modelo, pode ser mais fácil de entender e de lembrar. Note que um modelo não é autoexplicativo e precisa de informações extras – ou seja, uma legenda, exemplos, cenários etc.

- O foco em um único aspecto reduz a carga cognitiva necessária para compreender os requisitos modelados.

  Como um modelo tem um propósito específico e uma quantidade reduzida de informações, a compreensão da realidade modelada pode exigir menos esforço.

- As linguagens de modelagem de requisitos têm uma sintaxe restrita que reduz possíveis ambiguidades e omissões.

  Como a linguagem de modelagem (sintaxe e semântica) é mais simples – ou seja, número limitado de elementos de notação e regras de linguagem mais rígidas em comparação com a linguagem natural – o risco de confusão e omissões é menor.

- Maior potencial para análise e processamento automatizado dos requisitos.

  Como uma linguagem de modelagem é mais formal (número limitado de elementos de notação e regras de linguagem mais rígidas) do que uma linguagem natural, ela se presta melhor para automatizar a análise ou o processamento de requisitos.

Apesar das grandes vantagens de visualizar os requisitos com modelos, os modelos também têm suas limitações.

- Manter modelos que se concentram em diferentes aspectos consistentes entre si é um desafio.

  Se forem usados vários modelos para descrever os requisitos, é importante manter estes modelos consistentes uns com os outros. Isto requer muita disciplina e coordenação entre os modelos.

- Informações de diferentes modelos precisam ser compiladas para se obter um entendimento casual.

Se forem utilizados vários modelos, todos eles devem ser compreendidos para permitir uma boa compreensão dos requisitos.

- Os modelos se concentram principalmente nos requisitos funcionais.

  Os modelos para descrever os requisitos de qualidade e de restrições são limitados ou inexistentes e para contextos específicos. Estes tipos de requisitos devem então ser fornecidos em linguagem natural junto com os modelos – por exemplo, como um produto de trabalho separado.

- A sintaxe restrita de uma linguagem de modelagem gráfica implica que nem todas as informações relevantes podem ser expressas em um modelo.

  Como um modelo é feito para um propósito e contexto específicos, nem sempre é possível registrar todos os requisitos no modelo ou em vários modelos. Os requisitos que não podem ser expressos em modelos são adicionados ao modelo como requisitos de linguagem natural ou como um produto de trabalho separado.

Portanto, requisitos modelados devem ser sempre acompanhados por linguagem natural [Davi1995].

3.4.1.4 Aplicação de modelos de requisitos

Como indicado nas seções anteriores, existem modelos comuns para vários contextos. Por exemplo, na arquitetura, você tem desenhos de construção, diagramas de tubulação, diagramas elétricos etc., para expressar as especificações de um edifício. Em outros contextos – por exemplo, o desenvolvimento de software – existem linguagens de modelagem que são úteis nestes tipos de contexto. Um aspecto importante na aplicação de modelos é utilizar modelos que são comuns no contexto ou que tenham sido especialmente desenvolvidos para um contexto específico.

Várias linguagens de modelagem, por exemplo, UML [OMG2017] ou BPMN [OMG2013], foram padronizadas. Quando os requisitos são especificados em uma linguagem de modelagem não-padrão, a sintaxe e a semântica do idioma devem ser explicadas ao leitor – por exemplo, através de uma legenda.

Os modelos são usados para descrever os requisitos a partir de uma certa perspectiva. No desenvolvimento de sistemas, os requisitos funcionais são categorizados nas seguintes perspectivas (ver também a Seção 3.1.4):

- Estrutura e dados

  Modelos que focam nas propriedades estruturais estáticas de um sistema ou domínio

- Função e fluxo

  Modelos que focam na sequência de ações necessárias para produzir os resultados esperados de determinadas entradas ou nas ações necessárias para executar um processo (negócio), incluindo o fluxo de controle e dados entre as ações e o responsável por qual ação

- Estado e comportamento

  Modelos que focam no comportamento de um sistema ou no ciclo de vida de objetos de negócio em termos de reações dependentes do estado aos eventos ou da dinâmica da interação de componentes

A natureza do sistema sendo modificado ou construído indica quais dos modelos a utilizar. Por exemplo, se a natureza do sistema é processar informações e relacionamentos, então espera-se que haja muitos requisitos funcionais que descrevem estas informações e estes relacionamentos. Como resultado, usamos uma linguagem de modelagem correspondente que se presta à modelagem de dados e sua estrutura.

Naturalmente, um sistema consistirá de uma combinação das perspectivas acima. Segue-se que um sistema precisa ser modelado a partir de múltiplas perspectivas. As Seções 3.4.3 a 0 elaboram com mais detalhes os diferentes modelos para cada perspectiva.

Antes que os requisitos sejam levantados e documentados – por exemplo com modelos – um inventário é feito de objetivos e do contexto. Estes também podem ser modelados, ver Seções 3.4.6 respectivamente 3.4.2.

A aplicação de modelos nos ajuda principalmente das seguintes maneiras:

- *Especificar* requisitos (principalmente funcionais) parcial ou completamente, como um meio de substituir os requisitos representados textualmente
- *Decompor* uma realidade complexa em aspectos bem definidos e complementares; cada aspecto sendo representado por um modelo específico, ajuda-nos a compreender a complexidade da realidade
- *Parafrasear* requisitos textuais de forma a melhorar a sua compreensibilidade, em particular no que diz respeito às relações entre eles
- *Validar* requisitos textuais com o objetivo de descobrir omissões, ambiguidades e inconsistências

A modelagem dos requisitos também ajuda na estruturação e análise do conhecimento. Você pode usar diagramas para estruturar seus próprios pensamentos para obter uma melhor compreensão do sistema e de seu contexto.

3.4.1.5 Aspectos de qualidade de um modelo de requisitos

Esta é uma seção suplementar para a qual não haverá perguntas no exame de CPRE Foundation Level.

Uma parte substancial dos modelos de requisitos são diagramas ou representações gráficas. A qualidade do modelo de requisitos é determinada pela qualidade dos diagramas individuais e de suas relações mútuas. Por sua vez, a qualidade dos diagramas individuais é determinada pela qualidade dos elementos do modelo dentro dos diagramas.

A qualidade dos modelos de requisitos e dos elementos do modelo pode ser avaliada de acordo com três critérios [LiSS1994]:

- Qualidade sintática
- Qualidade semântica
- Qualidade pragmática

A qualidade sintática expressa até que ponto um único elemento de modelagem (gráfico ou textual), diagrama de requisitos ou modelo de requisitos está de acordo com as especificações sintáticas. Se, por exemplo, um modelo que descreve os requisitos como um modelo de classes contém elementos de modelagem que não fazem parte da sintaxe, ou os elementos do modelo são mal utilizados, então isso diminuirá a qualidade sintática do modelo. Um stakeholder deste modelo – por exemplo, um testador – pode interpretar mal as informações que são representadas pelo modelo. Isto pode eventualmente levar a casos de teste inadequados.

As ferramentas de modelagem de requisitos oferecem facilidades para a verificação da qualidade sintática dos modelos.

A qualidade semântica expressa a medida em que um único elemento de modelagem (gráfico ou textual), o diagrama de requisitos ou o modelo de requisitos representa correta e completamente os fatos.

Assim como na linguagem natural, a semântica dá sentido às palavras. Se um termo pode ter significados diferentes ou se há vários termos que significam a mesma coisa, isto pode levar a erros de comunicação. O mesmo se aplica à semântica dos elementos de modelagem. Se os elementos de modelagem forem mal interpretados ou aplicados de forma incorreta, o modelo pode ser mal interpretado.

A qualidade pragmática expressa até que ponto um único elemento de modelagem (gráfico ou textual), o diagrama de requisitos ou o modelo de requisitos é adequado para o uso pretendido – isto é, se o grau de detalhe e nível de abstração é apropriado para o uso pretendido e se o modelo apropriado é selecionado em relação ao domínio ou contexto. Isto pode ser avaliado se a finalidade e os stakeholders do diagrama forem conhecidos. As versões intermediárias do modelo podem ser submetidas aos stakeholders para validar se os diagramas se adequam ao seu propósito.

Durante a validação dos requisitos, a qualidade dos diagramas de modelagem utilizados é avaliada para garantir que esses diagramas se ajustem ao propósito e utilidade pretendidos.

3.4.1.6 O melhor dos dois mundos

Como explicado na seção anterior, os requisitos que são expressos de forma textual ou visual/gráfica (isto é, através de modelos de requisitos) têm suas vantagens e desvantagens. Utilizando tanto representações textuais quanto gráficas dos requisitos, podemos aproveitar o poder e os benefícios de ambas as formas de representação.

Acrescer um modelo com requisitos textuais adiciona mais significado ao modelo. Outra combinação útil é que podemos vincular requisitos de qualidade e de restrições a um modelo ou elemento de modelagem específico. Isto fornece uma visão mais completa de requisitos específicos.

O uso de modelos também pode suportar os requisitos textuais. A adição de modelos e imagens aos requisitos textuais apoia sua melhor compreensão e visão geral.

3.4.2 Modelar o Contexto do Sistema

O Capítulo 2, Princípio 4 introduz a noção de que os requisitos nunca vêm isoladamente e que o contexto do sistema, com seus sistemas, processos e usuários nele existentes, precisam ser considerados ao definir os requisitos para o novo sistema ou para o sistema alterado.

Os modelos de contexto especificam a incorporação estrutural do sistema em seu ambiente, com suas interações com os usuários do sistema, bem como com outros sistemas novos ou existentes dentro do contexto relevante. Um modelo de contexto não é uma descrição gráfica dos requisitos, mas é usado para revelar algumas das fontes dos requisitos. Figura 3.4 fornece um exemplo abstrato de um sistema e seu ambiente, com suas interfaces para os usuários do sistema e suas interfaces para outros sistemas. Assim, os diagramas de contexto ajudam a identificar as interfaces com usuários, bem como as interfaces com outros sistemas. Se o sistema interage com usuários, as interfaces de usuário devem ser especificadas em uma etapa posterior durante a ER.

Se o sistema interage com outros sistemas, estas devem ser definidas com mais detalhes em uma etapa posterior. As interfaces com outros sistemas podem já existir ou podem precisar ser desenvolvidas ou modificadas.

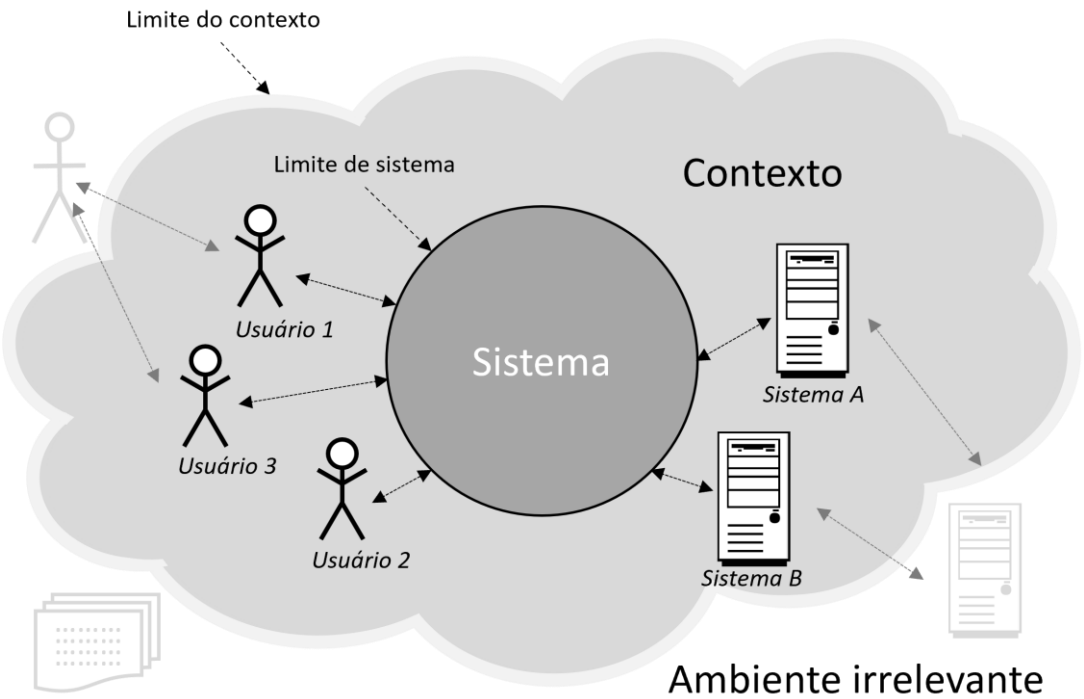


Figura 3.4 Um sistema em seu contexto

Mesmo que não exista uma linguagem de modelagem padronizada para modelos de contexto, os modelos de contexto são frequentemente representados por:

- Diagramas de fluxo de dados de análise estruturada [DeMa1978]
- Diagramas UML de casos de uso [OMG2017]
- Nota: um caso de uso UML existe em duas diferentes representações; o diagrama de caso de uso UML (ver Figura 3.6) e a especificação do caso de uso (Seção 3.4.2.2). Este capítulo se concentra na modelagem com os diagramas de caso de uso UML.
- Diagramas de blocos e linha [Glin2019]

No domínio da engenharia de sistemas, os diagramas de definição de blocos SysML [OMG2018] podem ser adaptados para expressar modelos de contexto utilizando blocos estereotipados para o sistema e os atores.

Nas duas subseções seguintes, introduzimos a notação de diagramas de fluxo de dados (DFD) e diagramas de caso de uso UML para modelar o contexto de um sistema. Estes dois exemplos não descrevem o contexto completamente, mas enfatizam o contexto a partir de um ponto de vista específico.

3.4.2.1 Diagrama de Fluxo de Dados

O contexto do sistema pode ser visto a partir de diferentes perspectivas. A análise estruturada dos sistemas [DeMa1978] fala sobre o diagrama de contexto. Este diagrama é um diagrama de fluxo de dados especial (DFD) onde o sistema é representado por um processo (o sistema). Figura 3.5 mosmostra um exemplo de um diagrama de contexto" – todo diagrama de contexto é DFD de nível zero.

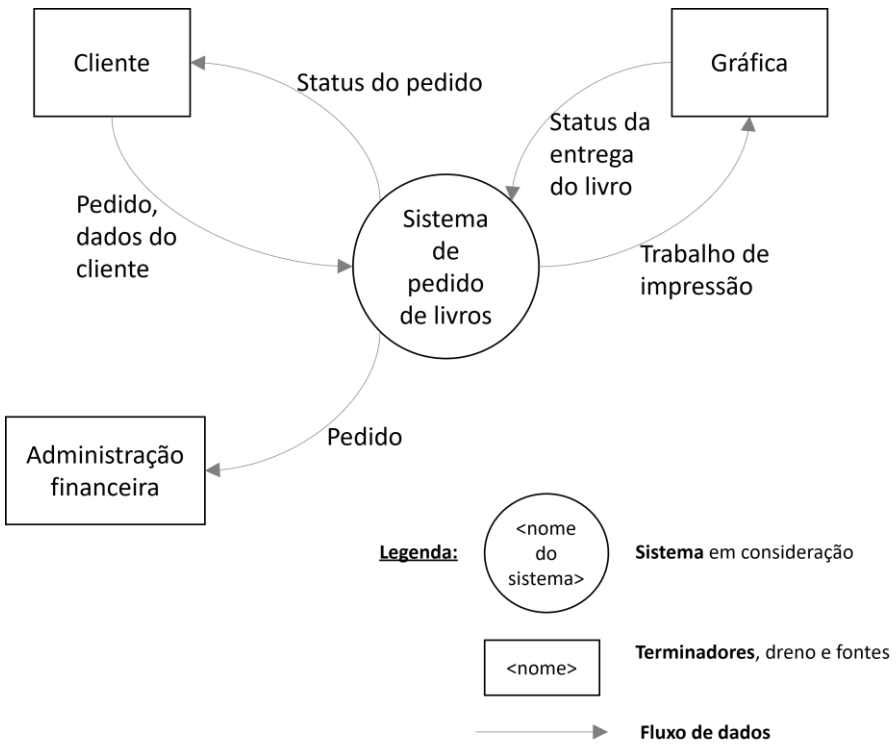


Figura 3.5 Exemplo de um diagrama de contexto usando DFD

O sistema é colocado de forma centralizada no modelo. Ele tem um nome claro para que os leitores saibam qual sistema está sendo considerado.

Os retângulos ao redor do sistema são terminadores: cliente, gráfica e administração financeira. Um terminador que fornece informações ou serviços para o sistema é chamado de *fonte*. Um terminador que retira informações ou serviços do sistema é chamado de *dreno*. Um terminador pode assumir qualquer função dependendo dos dados fornecidos ou recuperados, como o cliente no exemplo acima.

As setas no exemplo mostram como as informações dos terminadores fluem para o sistema (fonte) e do sistema para os terminadores (drenos). As setas recebem um nome lógico que descreve quais informações são transferidas. Detalhes irrelevantes são omitidos no nível do diagrama de contexto. O fluxo de informações entre o cliente e o sistema contém, por exemplo, *dados do cliente*. Quais informações (nome, data de nascimento, endereço eletrônico, número de telefone, endereço de entrega, endereço de faturamento etc.) compõem os *dados do cliente* ainda não precisam ser relevantes para este nível de abstração.

O fluxo de informações pode consistir em objetos tangíveis (materiais) e intangíveis (informações). Além disso, neste nível conceitual, não há referência (ainda) *ao como* — e-mail, website, formulário etc. — as informações são fornecidas.

Adicionar detalhes extras ao diagrama de contexto pode torná-lo mais claro para os stakeholders envolvidos e pode ajudar a melhorar o entendimento compartilhado. Estes detalhes precisam ser trabalhados para cada situação individual.

O uso de um diagrama de fluxo de dados para modelar o contexto de um sistema fornece algumas percepções sobre as interações do sistema com seu ambiente, por exemplo:

- As interfaces com pessoas, departamentos, organizações e outros sistemas no ambiente
- Os objetos (tangíveis e intangíveis) que o sistema recebe do ambiente
- Os objetos (tangíveis e intangíveis) que são produzidos pelo sistema e que são entregues ao ambiente

Um diagrama de fluxo de dados indica uma clara fronteira entre o sistema e seu ambiente. Os usuários e sistemas relevantes do ambiente são identificados durante a elicitação dos requisitos (Seção 4.1). Os diagramas de contexto DFD podem ajudar a estruturar o contexto para alcançar um entendimento compartilhado do contexto do sistema e dos limites do sistema.

3.4.2.2 Diagramas de Caso de Uso UML

Outra visão do contexto de um sistema pode ser alcançada a partir de uma perspectiva funcional. O diagrama de caso de uso UML é uma abordagem comum para modelar os aspectos funcionais de um sistema e os limites do sistema, juntamente com as interações do sistema com os usuários e outros sistemas. Os diagramas de casos de uso proporcionam uma maneira fácil de sistematicamente descrever as diversas funcionalidades dentro do

escopo definido, na perspectiva do usuário. Diferentemente dos diagramas de contexto da DFD, aqui o sistema é representado como uma grande caixa retangular.

Os casos de uso foram inicialmente propostos como um método para documentar as funcionalidades de um sistema em [Jaco1992]. Os casos de uso UML consistem em diagramas de caso de uso com especificações de caso de uso textual associadas (ver Seção 3.3.2). Uma especificação de caso de uso especifica cada caso de uso em detalhes, por exemplo, descrevendo as possíveis atividades do caso de uso, sua lógica de processamento, e as condições prévias e pós-condições da execução do caso de uso. A especificação dos casos de uso é essencialmente textual – por exemplo, através de templates de caso de uso, como recomendado em [Cock2001].

Como mencionado, um diagrama de caso de uso UML mostra as funcionalidades (casos de uso) do ponto de vista dos usuários diretos e outros sistemas que interagem com o sistema em consideração. O nome do caso de uso é frequentemente composto por um verbo e um substantivo. Isto dá uma breve descrição da função oferecida pelo sistema, como mostra o exemplo em Figura 3.6.

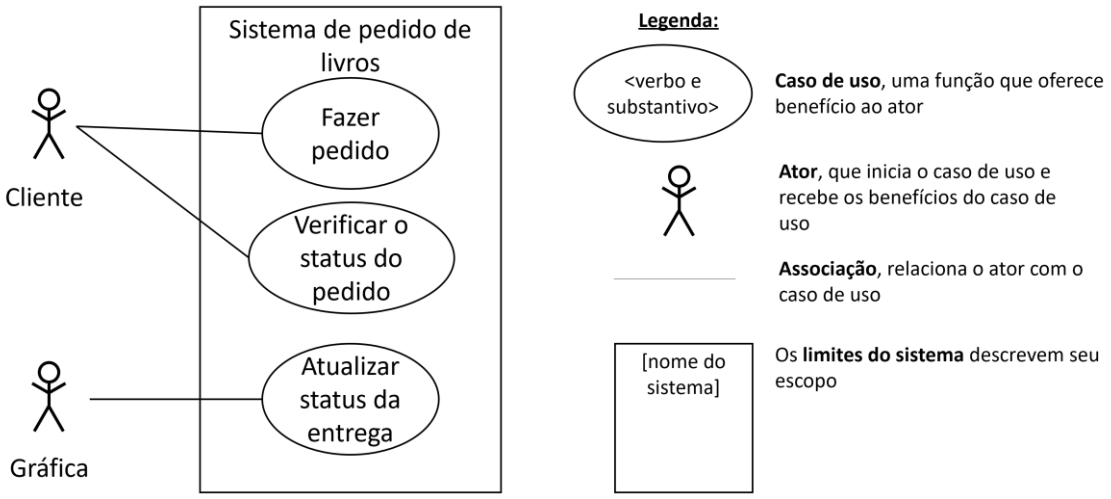


Figura 3.6 Exemplo de um diagrama de contexto usando um diagrama de caso de uso UML

Os atores são os usuários diretos ou sistemas que interagem com o sistema em consideração. O ator (usuário ou sistema) que inicia o caso de uso recebe o benefício que o caso de uso oferece (por exemplo, mostrando o status de um pedido ao cliente). A associação conecta o ator com o caso de uso relevante, mas não documenta nenhuma direção ou fluxo de dados (como é feito no DFD); ela expressa apenas que o ator recebe o benefício do caso de uso.

Um diagrama de caso de uso UML descreve a funcionalidade que o sistema oferece a seu ambiente. A separação entre a funcionalidade no sistema e os atores no contexto é visualizada com o limite do sistema (retângulo ao redor dos casos de uso, por exemplo, "sistema de pedido de livros"). Os diagramas de caso de uso permitem definir especificidades do limite do sistema e validar, em alto nível, se o escopo funcional do sistema está coberto.

Cada caso de uso também inclui uma especificação detalhada do caso de uso, documentando as pré-condições, gatilhos, ações, pós-condições, atores, e assim por diante. Os casos de uso são geralmente descritos usando um template (Seção 3.3). Se os cenários de um caso de uso se tornarem complexos ou grandes, a recomendação é visualizá-los através de diagramas de atividades UML, ver Seção 3.4.4.1. A especificação detalhada dos casos de uso não faz parte da modelagem de contexto e pode ser elaborada posteriormente, quando estas informações se tornarem relevantes.

3.4.3 Modelar Estruturas e Dados

Para requisitos funcionais na perspectiva de objetos de negócio (ver seção 3.1.4), diferentes modelos de dados. estão disponíveis. Um objeto (de negócio) pode ser um objeto tangível ou intangível, como uma bicicleta, pedal, buzina, mas também um pedido de treinamento, uma cesta de compras com produtos digitais, e assim por diante. Um objeto (de negócio) é "algo" no mundo real. Alguns (ou talvez todos) destes objetos (de negócio) são utilizados pelo sistema em consideração. O sistema usa esses objetos como entrada para processar, persistir e/ou entregar saída. Os modelos de dados são usados para descrever os objetos (de negócio) que devem ser conhecidos pelo sistema. Estes tipos de diagramas modelam o objeto, os atributos do objeto e as relações entre objetos. Por uma questão de simplicidade, nos referimos a modelagem de estrutura e dados – estes, entretanto, representam estruturas de informação entre objetos (de negócio) do mundo real.

Diversos modelos comuns para representação de estrutura e dados são:

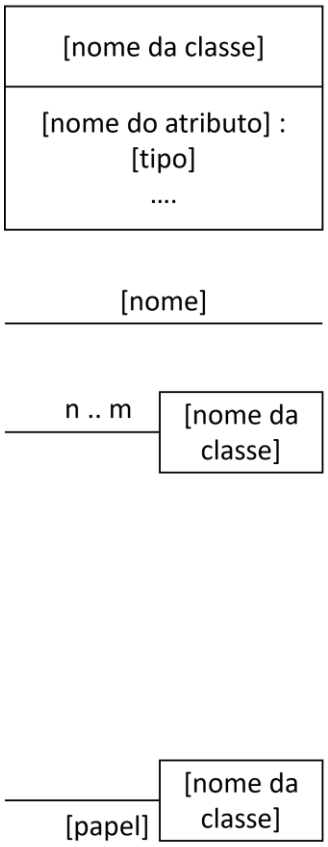
- Diagramas de Entidade e Relacionamentos (DER) [Chen1976]
- Diagramas de Classe UML [OMG2017]. Veja a Seção 3.4.3.1
- Diagramas de Definição de Blocos SysML [OMG2018]. Veja a Seção 3.4.6.2

Para explicar o conceito de estrutura de modelagem e dados, este capítulo utiliza o diagrama de classes UML como exemplo. UML, abreviação de Unified Modeling Language, consiste em um conjunto integrado de diagramas. Este conjunto de diagramas é uma coleção de melhores práticas de engenharia e tem se mostrado bem-sucedido na modelagem de sistemas complexos e grandes. A UML foi projetada por Grady Booch, James Rumbaugh e Ivar Jacobson na década de 1990 e é uma linguagem de modelagem padronizada desde 1997. Se desejar aprofundar ou usar um modelo diferente, leia a literatura mencionada e pratique com a linguagem de modelagem desejada.

3.4.3.1 Diagramas de Classe UML

UML é uma coleção de diferentes modelos que podem ser usados para descrever um sistema. Um desses modelos é o diagrama de classes. Um diagrama de classes retrata um conjunto de classes e associações entre elas. Apresentaremos apenas os elementos comuns e simples de notação deste modelo. Se for desejada maior profundidade, nos referimos à literatura ou ao CPRE Advanced Level Requirements Modeling.

Na visão geral abaixo você encontrará os elementos de notação mais comuns.



Uma **classe** descreve um conjunto de objetos (materiais ou imateriais) que têm estrutura, comportamento e relacionamentos similares.

Um **atributo** descreve uma propriedade da classe. O atributo é expresso como um certo **tipo** que restringe os valores que podem ser assinalados ao atributo.

Uma **associação** relaciona duas classes entre si.

As **multiplicidades** definem quantas instâncias da classe na extremidade correspondente da associação podem participar da associação a uma dada instância da classe na outra extremidade da associação. Onde e alguns exemplos:
0 .. 1 (zero ou uma vez)
0 .. * (zero ou mais vezes)
1 .. * (uma ou mais vezes)
7 (exatamente sete vezes)
1 (valor padrão, exatamente uma vez)

O **papel** define a função que um objeto da classe desempenha na associação

Figura 3.7 Subconjunto dos elementos de modelagem dos diagramas de classe UML

Em um modelo de classe, você encontrará os conceitos e termos que são relevantes no domínio. Estes conceitos incluem uma definição clara que está incluída no glossário. Com o uso de modelos de dados, o glossário é ampliado com informações sobre a estrutura e coerência dos termos e conceitos. Uma clara definição e coerência dos termos utilizados evita erros de comunicação sobre o assunto em consideração.

Figura 3.8 mostra um modelo simplificado do sistema de encomenda de livros (ver exemplos do contexto na seção 3.4.2). As informações estáticas que o sistema precisa para realizar suas funções – encomendando um livro – são modeladas.

Um cliente encomenda um livro e, portanto, as informações persistem para as classes *Cliente*, *Pedido* e *Livro*. Um cliente pode fazer um pedido e, portanto, existe uma relação (associação) entre o *cliente* e o *pedido*. Um cliente pode fazer vários pedidos ao longo do tempo e só se torna um cliente se ele fizer uma encomenda. Estas informações determinam a multiplicidade: 1 cliente faz 1 uma ou mais pedidos.

O fato de um cliente poder pedir um livro significa que existe também uma relação entre as classes *Pedido* e *Livro*. Para manter o exemplo simples, aqui, o cliente pode pedir apenas um livro de cada vez. Além disso, cada pedido deve conter pelo menos um livro. Um pedido sem livro não é um pedido.

Na classe *Livro,* o atributo *em Estoque* é mantido. Informações como "se o estoque não for suficiente para atender o pedido, então um trabalho de impressão é enviado para gráfica"

não pode ser modelado. Este é um tipo de informação que não pode ser modelado em um diagrama de classes porque descreve uma certa funcionalidade do sistema. Estas informações fazem parte dos requisitos e devem ser documentadas em outro produto de trabalho. Ele pode ser adicionado como um requisito textual que acompanha o diagrama de classes, ou ser modelado com outro diagrama – por exemplo, um diagrama de atividades UML (veja a Seção 3.4.4.1).

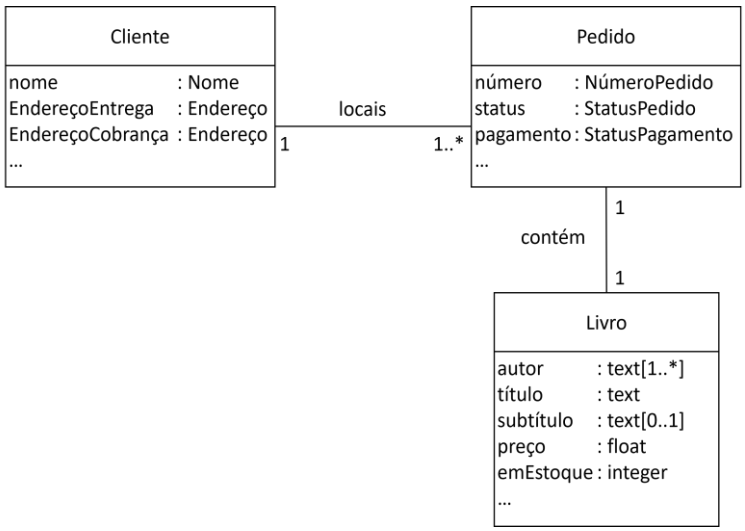


Figura 3.8 Exemplo de um diagrama de classes UML simples

3.4.4 Modelar Função e Fluxo

A função e o fluxo descrevem como o (sub)sistema deve transformar entrada em saída. Podemos visualizar este tipo de requisito com modelos que descrevem a função e o fluxo.

Ao contrário da modelagem de dados, que precisa essencialmente de apenas um tipo de diagrama, função e fluxo podem ser vistos de diferentes ângulos. Dependendo das necessidades dos stakeholders para dar o próximo passo no processo de desenvolvimento, mais de um modelo pode ser necessário para documentar os requisitos sobre função e fluxo.

Alguns modelos comuns para representar a função e o fluxo são:

- Diagramas de Caso de Uso UML [OMG2017]. Veja a Seção 3.4.2.2
- Diagrama de Atividade UML [OMG2017]. Veja a Seção 3.4.4.1
- Diagrama de Fluxo de Dados (DFD)[DeMa1978]. Veja a Seção 3.4.2.1
- Diagramas Domain Storytelling [HoSch2020]. Veja a Seção 3.4.6.3
- Business Process Model & Notation (BPMN) [OMG2013].

Excursão: Modelos de processo BPMN são usados para descrever processos de negócio ou processos técnicos. O BPMN é frequentemente usado para expressar modelos de processos negociais.

Para explicar o conceito de modelar função e fluxo, limitamos esta seção a alguns exemplos de diagramas UML. Se for desejado mais profundidade ou um modelo diferente, consulte a literatura referenciada e pratique com a linguagem de modelagem relevante.

3.4.4.1 Diagrama de atividade UML

Modelos de atividade são usados para especificar as funções do sistema. Fornecem elementos para modelar as ações e fluxo de controle entre as ações. Os diagramas de atividades também podem expressar o responsável pela ação. Os elementos avançados de modelagem (não cobertos por este guia de estudo) fornecem os meios para modelagem do fluxo de dados.

Um diagrama de atividades UML expressa o fluxo de controle de atividades de um (sub)sistema. Pensar em fluxos originou-se da visualização do código de programas com fluxogramas (de acordo com [DIN66001], [ISO5807]). Isto ajudou os programadores a conceber e compreender estruturas e fluxos complexos em programas. Com a introdução da UML [OMG2017], foi introduzido um modelo para visualização de atividades e ações a partir de uma perspectiva funcional.

Na visão geral abaixo você encontrará os elementos básicos de notação.

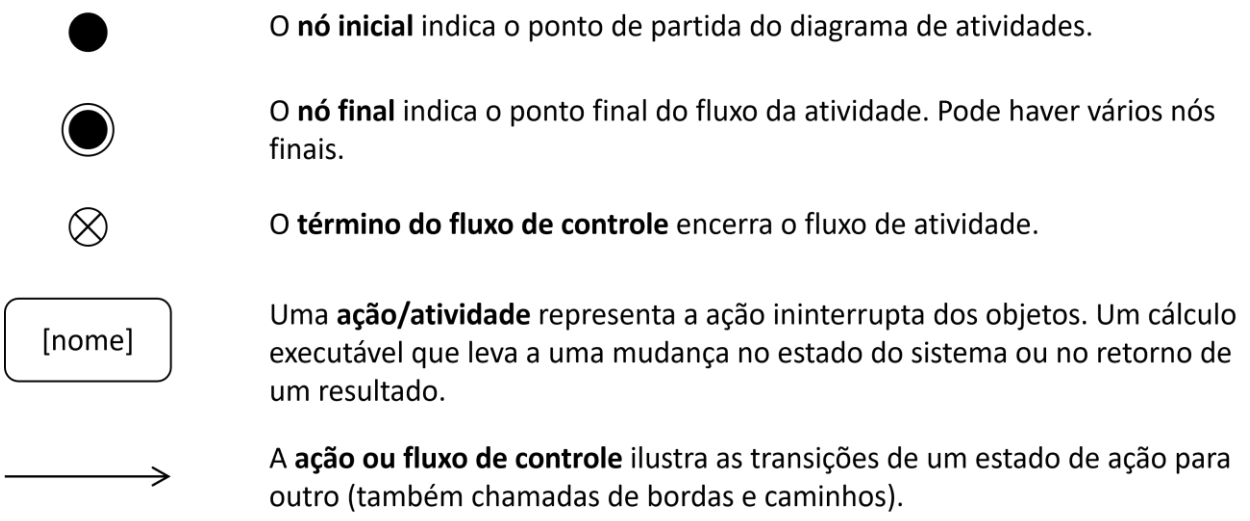
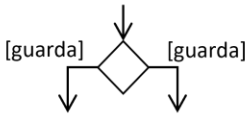
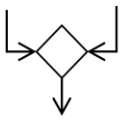


Figura 3.9 Elementos de notação básicos dos diagramas de atividades UML

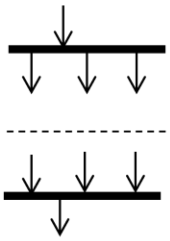
Com este conjunto de elementos de notação básicos, você pode montar um diagrama de atividade sequencial simples. Se for necessário mais controle, o modelo pode ser ampliado com fluxos de decisões e concorrentes de atividades usando os elementos de notação abaixo.



O **nó de decisão** ramifica o fluxo de atividade com base em uma guarda. As alternativas de saída devem ser rotuladas com uma condição ou expressão de guarda. A guarda deve ser verdadeira antes da mudança para o fluxo apropriado.



O **nó de junção** reúne múltiplos fluxos que não são concorrentes.



Barra de sincronização

Uma **barra de bifurcação** é utilizada para dividir um único fluxo de entrada em vários fluxos concorrentes.

Um **barra de junção** une vários fluxos concorrentes de volta em um único fluxo de saída. A ação após a barra de junção é executada após todas as ações paralelas terem sido executadas.

As barras de junção e bifurcação são usadas em conjunto, portanto, são frequentemente referidas como sincronização.

Figura 3.10 Fluxos de decisão e sincronização em diagramas de atividades UML

Os diagramas de atividades podem ser usados para especificar detalhadamente a lógica de processamento de cenários de casos de uso (ver Seção 3.3.2). Os diagramas de atividades são criados para visualizar os fluxos, que são processos com atividades e lógica de processamento. Enquanto o diagrama permanecer compreensível, o fluxo principal pode ser modelado em conjunto com os fluxos alternativos e os fluxos de exceção como parte do mesmo diagrama.

Figura 3.11 dá um exemplo simples do sistema de pedido de livros. Este fluxo de ações simplificado começa quando o cliente envia seu pedido. Primeiro, as informações do *pedido* do *Cliente* são validadas para determinar se todas as informações (necessárias) foram fornecidas. Se a informação do *pedido* ou *cliente* for inválida (incorreta ou insuficiente), então uma notificação é enviada ao cliente e o processo do pedido é cancelado. O fluxo principal é que o *Pedido* e as informações do *Cliente* sejam válidas. O cenário em que o *Pedido* ou informação do *Cliente* é inválido é chamado de um fluxo de exceção e lida funcionalmente com uma condição de falha no processo.

Se a informação do *Pedido* e *do Cliente* estiver correta, o estoque é verificado. Se há um número suficiente de produtos em estoque, o *Pedido* é separado e enviado ao *Cliente*. Um fluxo alternativo é iniciado se houver produtos insuficientes em estoque. Um pedido de impressão é enviado à *Gráfica* e uma notificação para uma nova entrega é enviada ao *Cliente*.

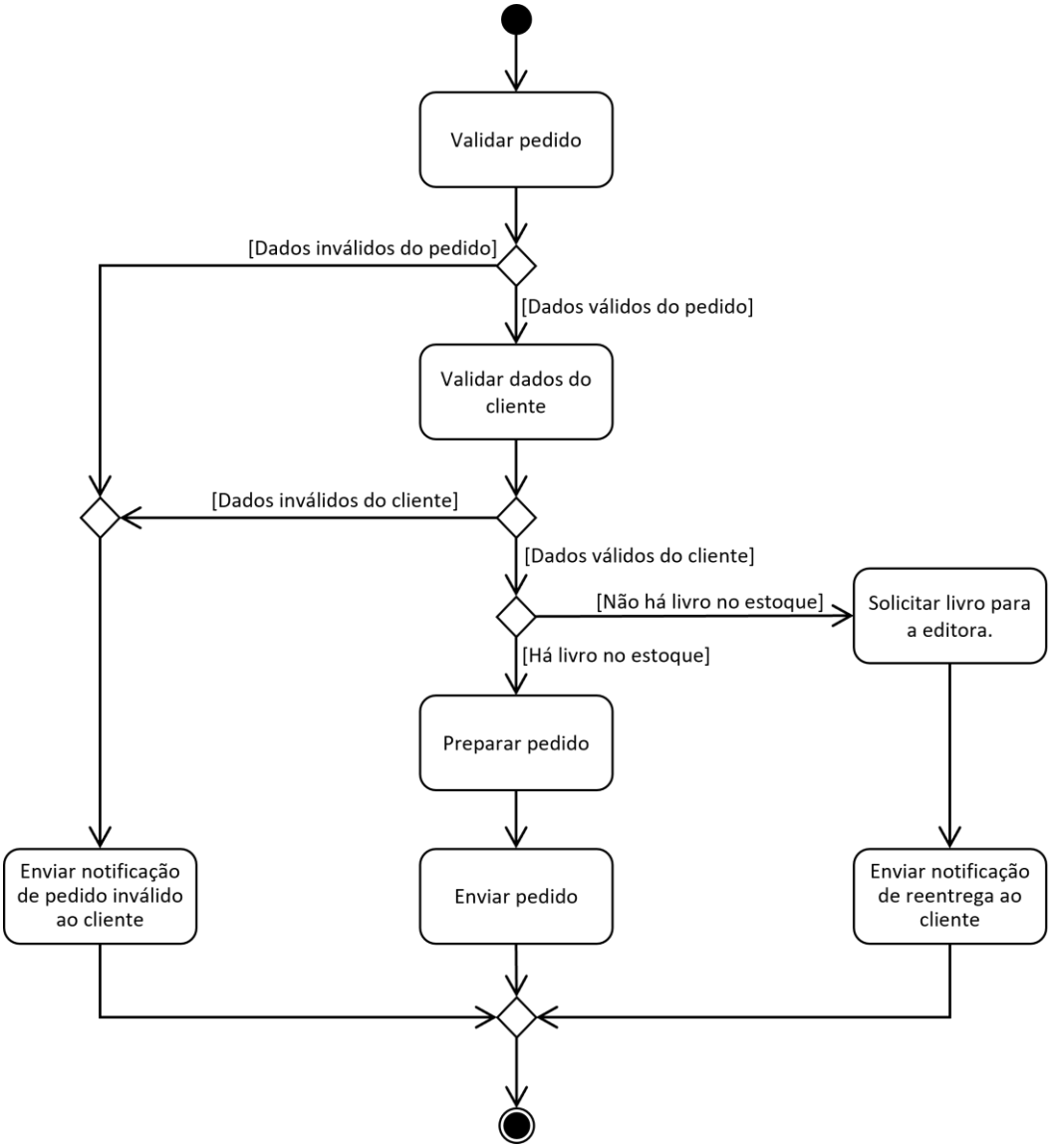


Figura 3.11 Exemplo de um diagrama de atividades UML

Dentro do sistema de pedidos de livros, há também outros fluxos que são separados do processo de pedido e entrega. Por exemplo, os processos de pagamento, reenvio e faturamento têm fluxos separados para permitir uma clara separação de preocupações. Se, por exemplo, for tomada a decisão de não manter mais nenhum produto em estoque, então o processo de pedido e entrega ainda se aplica. Se forem necessárias mudanças neste fluxo, estas mudanças podem não afetar os outros fluxos. Esta decomposição de funcionalidade ajuda a manter as coisas simples e claras.

3.4.5 Modelar Estado e Comportamento

Os requisitos funcionais que descrevem o comportamento, estados e transições de um (sub)sistema ou de um objeto de negócio são requisitos na perspectiva estado e comportamento. Um exemplo de um estado do sistema é *Ligado*, *em Espera* ou *Desligado*. Um objeto de negócio pode ter um ciclo de vida que passa por uma série de estados prescritos. Por exemplo, um objeto de negócio *Ordem* pode estar nos seguintes estados: *Solicitado*, *Validado*, *Pago*, *Enviado* e *Concluído*.

Uma técnica amplamente utilizada para descrever o comportamento de um sistema são os Statecharts [Hare1988]. Statecharts são máquinas de estado com estados decompostos hierarquicamente e/ou ortogonalmente. As máquinas de estado, incluindo Statecharts, podem ser expressas na linguagem de modelagem UML [OMG2017] com diagramas de máquina de estado (também chamados de diagramas de estado).

Os diagramas de estado descrevem as máquinas de estado finitas. Isto significa que estes sistemas eventualmente atingem um estado final. Um diagrama de estados mostra os estados que o sistema ou um objeto pode assumir. Também indica como mudar de estado – isto é, a transição de estado. Um sistema faz pouco por si só. Mudar de estado requer um gatilho do sistema ou do ambiente do sistema.

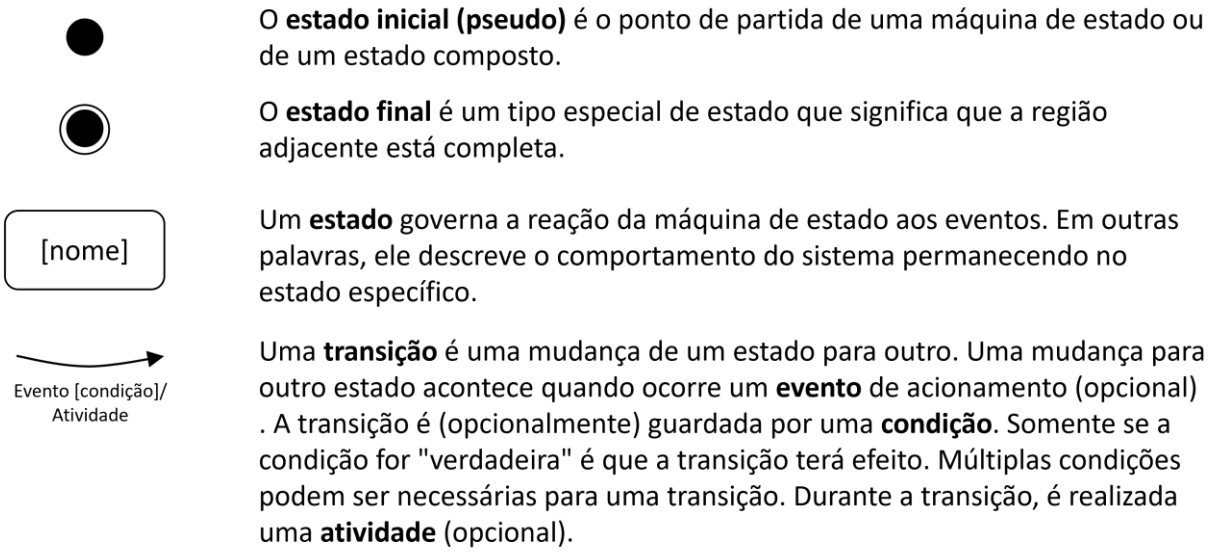
Os modelos comuns para representar comportamentos e estados incluem:

- Statecharts [Hare1988]
- Diagrama de estados UML [OMG2017]

3.4.5.1 Diagrama de estados UML

Para explicar o conceito de modelagem de comportamento e estado, este capítulo usa o diagrama de estados UML como exemplo. Se for desejado mais profundidade ou um modelo diferente, consulte a literatura referenciada e pratique com a linguagem de modelagem relevante.

Na visão geral abaixo você encontrará os elementos básicos de notação.



O **estado inicial (pseudo)** é o ponto de partida de uma máquina de estado ou de um estado composto.

O **estado final** é um tipo especial de estado que significa que a região adjacente está completa.

Um **estado** governa a reação da máquina de estado aos eventos. Em outras palavras, ele descreve o comportamento do sistema permanecendo no estado específico.

Uma **transição** é uma mudança de um estado para outro. Uma mudança para outro estado acontece quando ocorre um **evento** de acionamento (opcional) . A transição é (opcionalmente) guardada por uma **condição**. Somente se a condição for "verdadeira" é que a transição terá efeito. Múltiplas condições podem ser necessárias para uma transição. Durante a transição, é realizada uma **atividade** (opcional).

Figura 3.12 Elementos básicos de notação de diagrama de estados UML

Como discutido no início da seção, um diagrama de estados pode esclarecer os estados que um objeto pode tomar. Vemos aqui uma oportunidade de visualizar informações adicionais (e parcialmente redundantes) de um objeto. Imagine que você compra um livro em um site e quer acompanhar o status de seu pedido. Um pedido é usado no mundo real e é modelado como um objeto de negócios em um diagrama de classes (ver Figura 3.8), muito provavelmente, com um *status de* atributo. O diagrama de classes indica que o atributo de *status* pode assumir um número limitado de valores, tais como *Solicitado*, *Pago*, *Entregue*, *Cancelado* e assim por diante. O diagrama de classes não descreve o pedido de possíveis mudanças de status. Um diagrama de classes também não descreve o comportamento do sistema em um determinado "status". Isto pode ser deixado claro com um diagrama de estado UML, por exemplo, que um pedido oferecido não pode ir diretamente ao status *Entregue* sem que o cliente tenha pago pelo pedido.

Figura 3.13 dá um exemplo de um diagrama de estado do sistema de pedidos de livros. No diagrama de classes (Figura 3.8) do sistema de encomenda de livros, um objeto *Pedido* é modelado. Este objeto tem um atributo de *status* que pode ter um número limitado de valores. Estes valores estão listados e explicados no diagrama de classes. O que um diagrama de classes não descreve é a sequência em que o pedido é processado. Um diagrama de estados visualiza os estados e as transições entre os estados, deixando claro qual é a sequência do status do pedido. O diagrama de estados mostra, por exemplo, que o

pedido não pode ser enviado antes de ser completamente separado (transição entre os estados *Separado* e *Enviado*). Além disso, se o pedido estiver no estado *Enviado,* o próximo estado só poderá ser *Pago*. A transição de *Enviado* para *Tratado* não é possível. Este diagrama também deixa claro que o pagamento acontece depois que o livro é enviado. Você pode perguntar aos stakeholders se é isto que precisam ou solicitaram.

Uma transição pode ocorrer para o mesmo status. Esta situação é visível no estado *Separado*. Enquanto o pedido não for separado completamente, ele permanece no mesmo estado para evitar enviar um pedido incompleto. Somente quando o pedido dor completamente separado é em seguida enviado ao cliente.

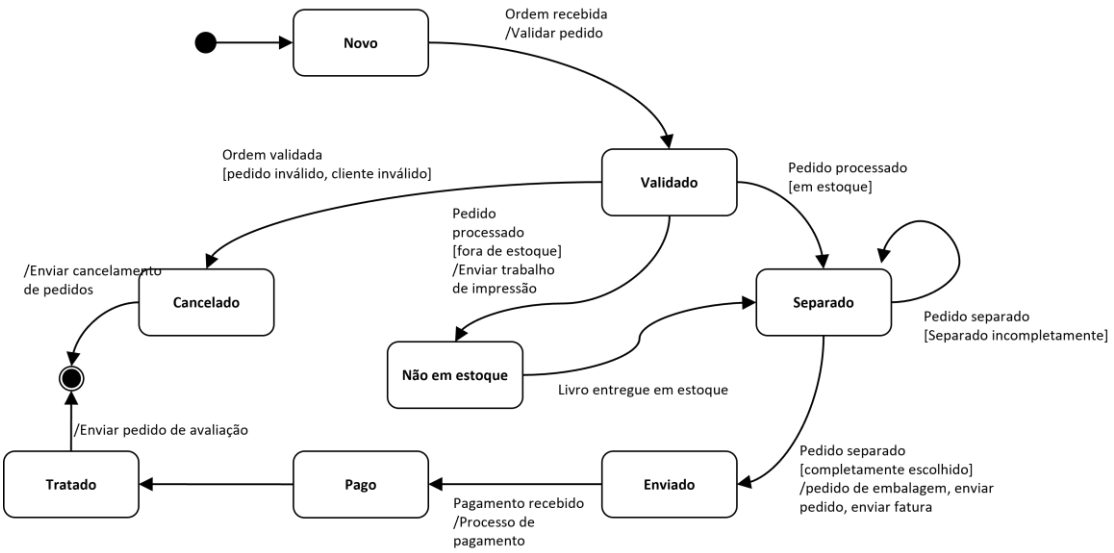


Figura 3.13 Exemplo de um diagrama de estado UML

Alguns meses após o lançamento do sistema de pedidos de livros, os clientes reclamaram que não tinham a capacidade de cancelar um pedido. Ficou acordado que um cliente poderia cancelar o pedido em qualquer estado do processo de pedido. Modelar este novo requisito significa que é necessária uma transição de cada estado para *Cancelado*. Isto pode tornar difícil a leitura do diagrama. Adicionar um requisito textual para descrever este comportamento pode ser uma maneira de manter o modelo simples para o público.

3.4.6 Modelos complementares

No nível fundamental CPRE, a compreensão e aplicação de modelos é restrita a tipos de modelos selecionados e importantes. Há outros tipos de modelos que são usados na Engenharia de Requisitos. As subseções a seguir fornecem alguns exemplos adicionais de modelos que são complementares e não serão questionados no exame de nível CPRE Foundation.

3.4.6.1 Modelar Objetivos

Os requisitos negociais descrevem um objetivo ou necessidade de negócio. Eles descrevem o resultado final que a solução deve encontrar e que problema (de negócio) é resolvido, ver Capítulo 2, Princípio 5. Para assegurar que o foco esteja na resolução do problema e que o esforço se concentre em agregar valor, os objetivos são cuidadosamente descritos. Na Engenharia de Requisitos, há várias maneiras de documentar objetivos. O mais comum é o uso de linguagem natural (Seção 3.2) ou templates (Seção 3.3). Formas de documentação baseadas em templates podem ser encontrados, por exemplo, em [Pich2010], [Pohl2010], ou [RoRo2012].

Há também algumas notações baseadas em modelos para a documentação de objetivos. A notação mais fácil e mais comum é o diagrama de Árvore AND/OR [AnPC1994]. Árvores AND/OR nos permitem documentar objetivos em diferentes níveis de detalhe e ligar as submetas com objetivos usando as relações AND e OR. Um relacionamento AND significa que todas as submetas precisam ser cumpridas para cumprir o objetivo. Um relacionamento OR é usado para expressar que pelo menos uma das submetas precisa ser cumprida para cumprir o objetivo.

Abordagens de modelagem mais elaboradas para metas podem ser encontradas em:

- Goal-oriented Requirements Language (GRL) [GRL2020]

  Esta é uma linguagem que suporta modelagem orientada a objetivos e raciocínio sobre requisitos, especialmente para lidar com requisitos não-funcionais.

- Knowledge aquisition in automated specification (ou Keep All Objectives Satisfied (KAOS)) [vLam2009]

  A KAOS é uma metodologia que contém modelagem de objetivos. Isto permite que os analistas construam modelos de requisitos e obtenham documentos de requisitos a partir dos modelos de objetivos do KAOS.

- A estrutura i* é um dos métodos de modelagem e raciocínio orientados a metas e agentes mais populares da área. O i* oferece suporte à criação de modelos que representam uma organização ou um sistema sociotécnico. [FLCC2016] fornece uma visão geral abrangente da estrutura i* e de seus aplicativos.

A documentação dos objetivos (em forma textual ou gráfica) é um importante ponto de partida para elicitar requisitos, referi-los à sua lógica e identificar suas fontes – como stakeholders – etc.

3.4.6.2 Diagramas de Definição de Blocos SysML

Systems Modeling Language (SysML) [OMG2018] é uma linguagem de modelagem de uso geral para aplicativos de engenharia de sistemas. A SysML é um dialeto da UML, que reutiliza e amplia partes da UML.

A SysML pode ser adotada para muitas finalidades diferentes. *Os diagramas de definição de blocos* em SysML são uma extensão do diagrama de classes UML. Eles podem, por exemplo,

ser adaptados para expressar diagramas de contexto usando blocos estereotipados para o sistema e os atores. Os diagramas de definição de blocos também podem ser usados para modelar a estrutura de um sistema em termos das entidades conceituais do sistema e das relações entre elas.

3.4.6.3 Diagramas domain storytelling

Os modelos de histórias de domínio podem ser usados para modelar a função e o fluxo, especificando histórias visuais sobre como os atores interagem com dispositivos, artefatos e outros itens em um domínio, tipicamente usando símbolos específicos do domínio [HoSch2020]. Eles são um meio para entender o domínio de aplicação no qual um sistema irá operar.

As técnicas devem ser executadas de forma muito simples para contar uma história; um quadro e notas adesivas podem ser suficientes. Reunir as partes interessadas relevantes que realmente sabem como o negócio funciona provoca uma discussão significativa ao contar histórias que ocorrem no domínio. A narração de histórias de domínio melhora o entendimento compartilhado de um processo de negócios e é usada para analisar e resolver problemas no domínio.

3.4.6.4 Diagrama de sequência UML

O diagrama de sequência UML é usado para representar a interação de comunicação entre parceiros e modela o aspecto dinâmico dos sistemas. O aspecto dinâmico dos sistemas que um diagrama de sequência descreve pode ser a função e o fluxo, bem como o estado e o comportamento. Portanto, um diagrama de sequência UML pode ser usado para diferentes finalidades.

Os parceiros de comunicação em diagramas de sequencia UML são atores, sistemas, componentes e/ou objetos dentro de um sistema. A interação exibe a sequência de mensagens (um cenário) entre esses parceiros de comunicação. A interação que ocorre entre os parceiros de comunicação realiza o propósito de um cenário, respectivamente (uma parte) de um caso de uso.

Na visão geral abaixo você encontrará os elementos básicos de notação.

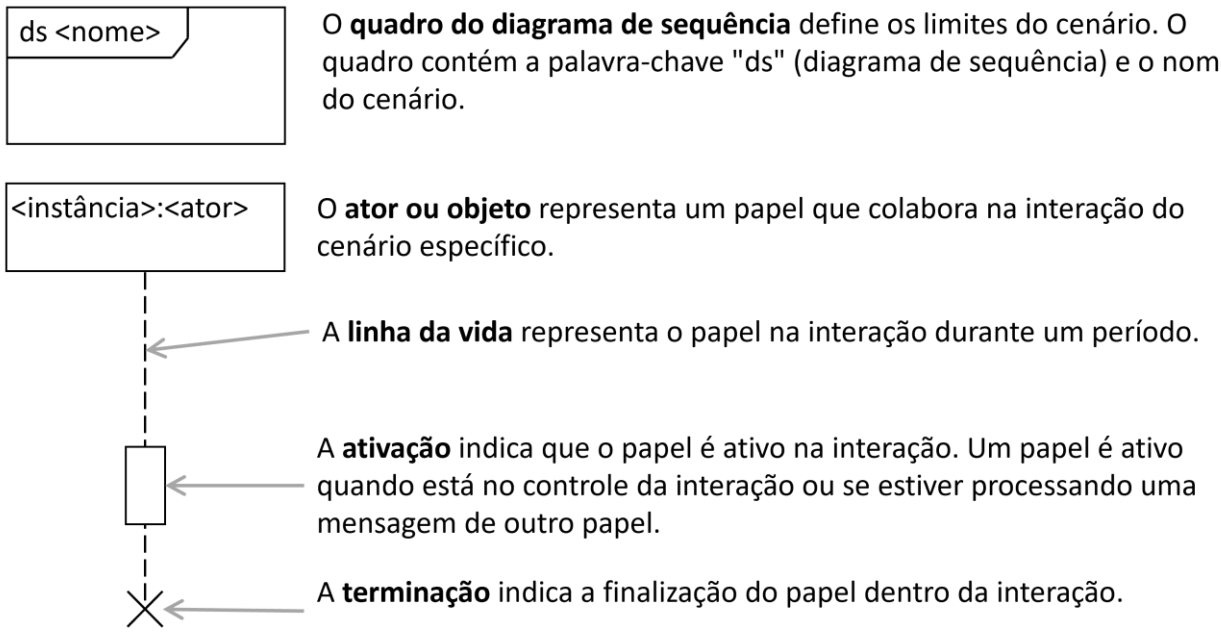


Figura 3.14 Elementos básicos de notação]do diagrama de sequência UML

Uma linha de vida em um cenário representa o papel no cenário, ou seja, a instância de um ator. Quando os diagramas de sequência são modelados, o nome da instância de um ator ou objeto é frequentemente omitido. Os papéis que participam da comunicação interagem uns com os outros através do envio de mensagens. Há dois tipos de mensagens que são usadas na interação.

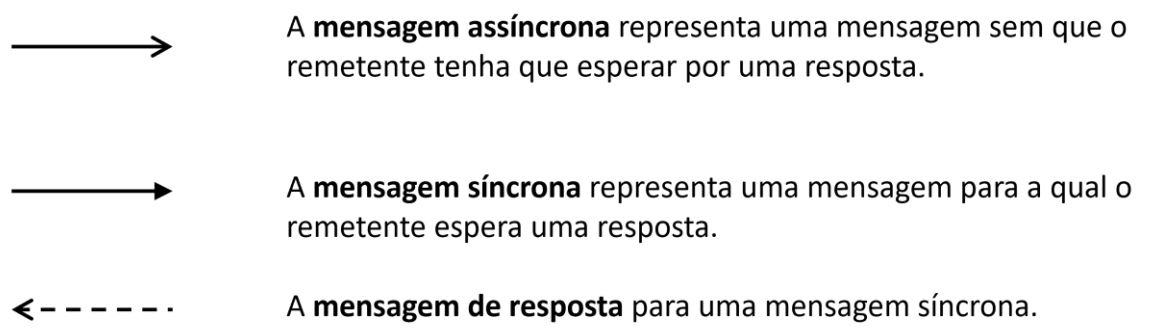


Figura 3.15 Elementos básicos de notação de mensagens no diagrama de sequência UML

Uma mensagem também pode ser enviada de ou para objetos fora do cenário. Isto é representado como um círculo preenchido. O remetente ou receptor deste tipo de mensagens pode ser desconhecido.

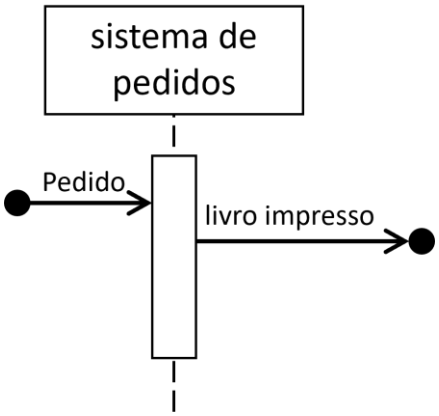


Figura 3.16 Mensagens de e para objetos fora do cenário

Figura 3.17 mostra um modelo do cenário no qual um *cliente* pede um livro que está esgotado. O *Cliente* faz um *Pedido*. Se o *Pedido* for inválido, uma notificação de que o *Pedido* foi cancelado é retornada. Se o *Pedido* for válido, o estoque é verificado e se um livro estiver fora de estoque, uma solicitação de impressão é enviada à *Gráfica*.

Essa é uma mensagem síncrona porque estamos aguardando o recebimento do livro – mesmo que leve algum tempo para imprimir o livro. É enviada uma notificação ao *Cliente* de que o livro está fora de estoque e será entregue posteriormente. O *Pedido* é desativado até que o livro seja entregue pela *Gráfica*.

Quando o livro é recebido da *impressora,* o *sistema de pedidos* é ativado novamente. O pedido é separado e enviado ao *Cliente*. Isto completa o *Pedido* e uma última notificação do status é enviada ao *Cliente*.

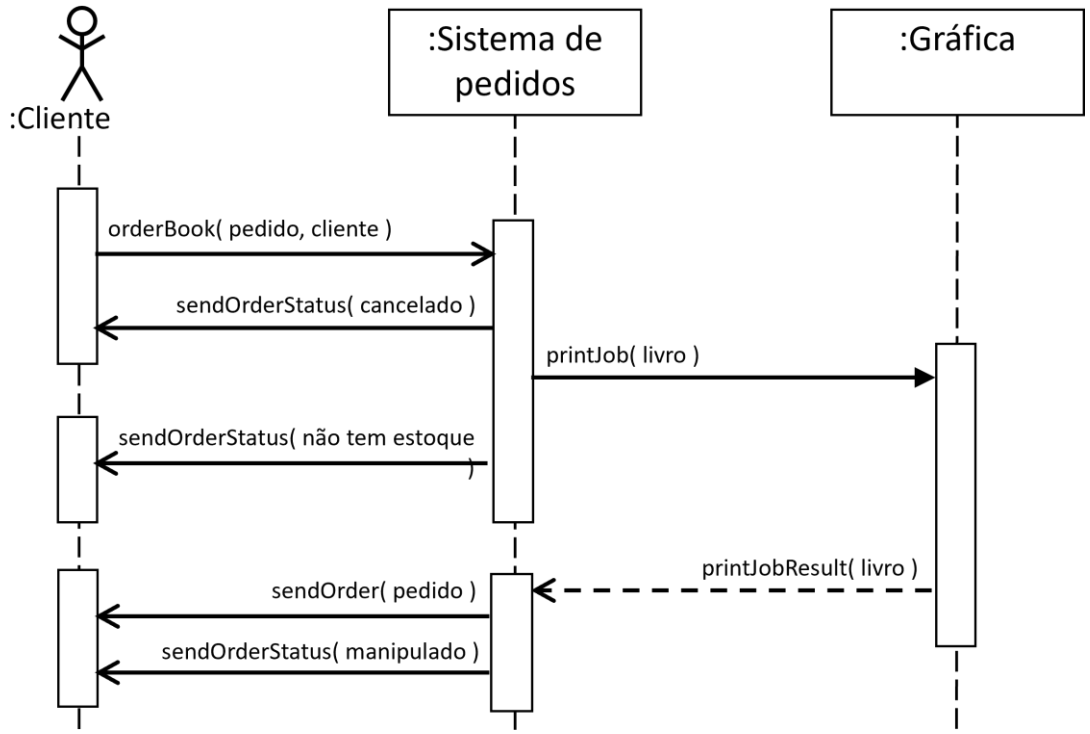


Figura 3.17 Exemplo de um diagrama de sequência UML

3.5 Glossários

Os glossários são um meio essencial para estabelecer um entendimento compartilhado da terminologia utilizada no desenvolvimento de um sistema: eles ajudam a evitar que as pessoas envolvidas como stakeholders ou desenvolvedores usem e interpretem os mesmos termos de maneiras diferentes.

Um bom glossário contém definições para todos os termos que são relevantes para o sistema, sejam eles termos específicos do contexto ou termos cotidianos que são usados com um significado especial no contexto do sistema a ser desenvolvido. Um glossário também deve definir todas as abreviações e acrônimos utilizados. Se houver sinônimos (ou seja, termos diferentes denotando a mesma coisa), eles devem ser marcados como tal. Homônimos (ou seja, termos idênticos que denotam coisas diferentes) devem ser evitados ou pelo menos marcados como tal no glossário.

Há algumas regras que orientam a criação, uso e manutenção do glossário em um projeto de desenvolvimento de sistema.

- *Criação e manutenção*. Para garantir que a terminologia definida no glossário seja consistente e sempre atualizada, é vital que o glossário seja gerenciado e mantido de forma centralizada durante todo o curso de um projeto, sendo uma pessoa ou um pequeno grupo responsável pelo glossário. Ao definir os termos, é importante que os stakeholders estejam envolvidos e concordem com a terminologia.
- *Utilização*. Para obter o pleno benefício de um glossário, seu uso deve ser obrigatório. Os produtos de trabalho devem ser verificados quanto ao uso adequado do glossário. Obviamente, isto significa que todos os envolvidos em um projeto devem ter acesso ao glossário.

Quando uma organização desenvolve sistemas inter-relacionados em múltiplos projetos, faz sentido criar um glossário no nível da organização para alcançar uma terminologia consistente entre os projetos.

A criação, manutenção e utilização de um glossário evita consistentemente erros e mal-entendidos com relação à terminologia utilizada. Trabalhar com glossários é uma boa prática na ER.

3.6 Documentos de Requisitos e Estruturas de Documentação

Não é suficiente trabalhar com requisitos no nível de requisitos individuais. Os requisitos devem ser reunidos e agrupados em produtos de trabalho adequados, sejam eles documentos de requisitos explícitos ou outras estruturas de documentação relacionadas à ER (tais como um backlog de produto).

Templates de documentos (ver Seção 3.3.3) podem ser usados para organizar tais documentos com uma estrutura bem definida, a fim de criar uma coleção de requisitos consistente e sustentável. Templates de Documentos estão disponíveis na literatura

[Vole2020], [RoRo2012] e em normas [ISO29148]. Os templates também podem ser reutilizados de projetos anteriores similares ou podem ser impostos por um cliente. Uma organização também pode decidir criar um template de documento como um padrão interno.

Um documento de requisitos também pode conter informações e explicações adicionais – por exemplo, um glossário, critérios de aceite, informações sobre o projeto ou características técnicas da implementação.

Os documentos de requisitos usados com frequência são:

- *Especificação de Requisitos dos Stakeholders*: os desejos e necessidades dos stakeholders que devem ser satisfeitos através da construção de um sistema, visto da perspectiva dos stakeholders. Quando clientes escrevem uma especificação de requisitos dos stakeholders, são chamadas de *Especificação de Requisitos dos Clientes*.
- *Especificação de Requisitos dos Usuários:* um subconjunto de uma especificação de requisitos dos stakeholders, abrangendo apenas os requisitos dos stakeholders que são usuários em potencial de um sistema.
- *Especificação de Requisitos do Sistema*: os requisitos para um sistema a ser construído e seu contexto para que ele satisfaça os desejos e necessidades de seus stakeholders.
- *Especificação de Requisitos de Negócio*: as metas, objetivos e necessidades negociais de uma organização que devem ser alcançados empregando um sistema (ou um conjunto de sistemas).
- *Documento de Visão*: uma imaginação conceitual de um sistema futuro, descrevendo suas principais características e como ele criará valor para seus usuários.

Estruturas alternativas de documentação frequentemente utilizadas:

- *Backlog do Produto*: uma lista priorizada de itens de trabalho, cobrindo todos os requisitos necessários e conhecidos para o produto
- *Backlog da Sprint*: um subconjunto selecionado de um Backlog de Produto com itens de trabalho que serão realizados na próxima iteração
- *Mapa de Histórias*: uma organização visual bidimensional de histórias de usuário de um backlog de produto em relação ao tempo e conteúdo

Não há documento – ou estrutura de documento – de requisitos padrão ou universal. Assim, os documentos ou estruturas de documentação não devem ser reutilizados de projetos anteriores sem reflexão.

- A escolha real depende de vários fatores, por exemplo:
- A abordagem de desenvolvimento escolhida
- O tipo de projeto e domínio (por exemplo, solução sob medida, desenvolvimento ou sustentação de produto)
- O contrato (um cliente pode prescrever o uso de uma determinada estrutura de documentação)
- O tamanho do documento (quanto maior o documento, mais estruturado deverá ser)

3.7 Protótipos na Engenharia de Requisitos

Os protótipos desempenham um papel importante tanto na engenharia quanto no design.

> Definição 3.5 Protótipo:
> 1. Na fabricação: Uma peça que é construída antes do início da produção em massa.
> 2. Na engenharia de software e sistemas: Uma realização preliminar e parcial de certas características de um sistema.
> 3. No desenho: Uma instância preliminar e parcial de uma solução de Design.

Os protótipos na engenharia de software e sistemas são utilizados para três propósitos principais [LiSZ1994]:

Protótipos exploratórios são usados para criar entendimento compartilhado, esclarecer requisitos ou validar requisitos em diferentes níveis de fidelidade. Tais protótipos constituem produtos de trabalho temporário que são descartados após o uso. Os protótipos exploratórios também podem ser usados como meio de especificação através de exemplos. Tais protótipos devem ser tratados como produtos de trabalho evolutivos ou duráveis.

Protótipos experimentais (também chamados de breadboards) são utilizados para explorar conceitos de soluções técnicas de projeto, em particular no que diz respeito à sua viabilidade técnica. Eles são descartados após o uso. Os protótipos experimentais não são utilizados na ER.

Protótipos evolutivos são sistemas-piloto que formam o núcleo de um sistema a ser desenvolvido. O sistema final evolui através da ampliação e melhoria progressiva do sistema-piloto em várias iterações. O desenvolvimento de sistemas ágeis frequentemente emprega uma abordagem de prototipação evolutiva.

Os Engenheiros de Requisitos utilizam principalmente protótipos exploratórios como um meio de elicitação e validação de requisitos. Na elicitação, os protótipos servem como um meio de especificação através de exemplos. Em particular, quando os stakeholders não expressam claramente o que querem, um protótipo pode demonstrar o que obteriam, o que os ajuda a definir seus requisitos. Na validação, os protótipos são um meio poderoso para validar a *adequação* (ver Seção 3.8) dos requisitos.

Os protótipos exploratórios podem ser construídos e utilizados com diferentes graus de fidelidade. Distinguimos entre wireframes, maquetes e protótipos nativos.

Wireframes (também chamados de protótipos de papel) são protótipos de baixa fidelidade construídos com papel ou materiais simples que servem principalmente para discutir e validar ideias de design e conceitos de interface com o usuário. Na prototipagem de sistemas digitais, os wireframes também podem ser construídos com programas Sketch ou

ferramentas específicas para wireframes. Entretanto, ao utilizar uma ferramenta de wireframe, é importante manter suas propriedades essenciais: ela pode ser construída rapidamente, modificada facilmente e ser rústica não se assemelhando a um produto final.

Maquetes são protótipos de média fidelidade. Ao especificar sistemas digitais, eles usam telas reais e fluxos de cliques, mas sem funcionalidade real. Eles servem principalmente para especificar e validar interfaces de usuário. As maquetes dão aos usuários uma experiência realista de como interagir com um sistema através de sua interface de usuário. Eles são normalmente construídos com ferramentas de prototipação especializadas.

Protótipos Nativos são protótipos de alta fidelidade que implementam partes críticas de um sistema de tal forma que os stakeholders possam usar o protótipo para ver se a parte prototipada do sistema funcionará e se comportará como esperado.

Eles servem tanto para a especificação por exemplos quanto para a validação abrangente dos requisitos críticos. Protótipos nativos também podem ser usados para explorar e decidir sobre *variantes de* requisitos para algum aspecto – por exemplo, duas formas diferentes possíveis de apoiar um determinado processo de negócio.

Dependendo do grau de fidelidade, os protótipos podem ser um produto de trabalho caro. Os Engenheiros de Requisitos têm que considerar a melhor relação entre o custo de construção e uso de protótipos e o benefício ganho em termos de facilidade de elicitação e na redução de risco de obtenção de requisitos inadequados ou mesmo errados.

3.8 Critérios de Qualidade para Produtos de trabalho e Requisitos

Obviamente, os Engenheiros de Requisitos devem se esforçar para escrever bons requisitos que atendam a determinados critérios de qualidade. A literatura e as normas da ER fornecem um rico conjunto de tais critérios de qualidade. Entretanto, não há um consenso geral sobre quais critérios de qualidade devem ser aplicados para os requisitos. O conjunto de critérios apresentados nesta subseção tem como objetivo fornecer uma prática comprovada em nível fundamental.

A moderna ER segue uma abordagem orientada a valor para os requisitos (ver Princípio 1 no Capítulo 2). Consequentemente, o grau em que um requisito atende a determinados critérios de qualidade deve corresponder ao valor criado pelo requisito. Isto tem duas consequências importantes:

- Os requisitos não têm que atender totalmente a todos os critérios de qualidade.
- Alguns critérios de qualidade são mais importantes do que outros.

Fazemos distinção entre critérios de qualidade para requisitos simples e critérios de qualidade para produtos de trabalho da ER, tais como documentos ou estruturas de documentos da ER.

Para requisitos individuais, recomendamos o uso dos seguintes critérios de qualidade:

- *Adequado*: o requisito descreve as reais e acordadas necessidades dos stakeholders.

- *Necessário*: o requisito faz parte do escopo relevante do sistema, o que significa que ele contribuirá para a realização de pelo menos um objetivo ou necessidade dos stakeholders.
- *Sem ambiguidade*: há um real entendimento compartilhado do requisito, o que significa que todos os envolvidos a interpretam da mesma forma.
- *Completo*: o requisito se basta, o que significa que não faltam partes necessárias para compreendê-lo.
- *Compreensível*: o requisito é compreensível para o público-alvo, o que significa que o público-alvo pode compreender totalmente o requisito.
- *Verificável*: o cumprimento do requisito por um sistema implementado pode ser verificado incontestavelmente (para que os stakeholders ou clientes possam decidir se um requisito é ou não cumprido pelo sistema implementado).

Adequação e Compreensibilidade são os critérios de qualidade mais importantes. Sem eles, um requisito é inútil ou até prejudicial, independentemente do cumprimento de todos os outros critérios. A verificabilidade é importante quando o sistema implementado deve passar por um procedimento formal de aceite.

Algumas pessoas usam *a exatidão* em vez da *adequação*. Entretanto, a noção de exatidão implica que existe um procedimento formal para decidir se algo está exato ou não. Como não existe um procedimento formal para validar um requisito documentado contra os desejos e necessidades que os stakeholders têm em mente, preferimos o termo adequação em vez de exatidão.

Para produtos de trabalho que cobrem múltiplos requisitos, recomendamos a aplicação dos seguintes critérios de qualidade:

- *Consistente*: não há dois requisitos, registrados em um único produto de trabalho ou em produtos de trabalho diferentes, que se contradigam.
- *Não redundante*: cada requisito é documentado apenas uma vez e não se sobrepõe a outro requisito.
- *Completo*: o produto de trabalho contém todos os requisitos relevantes (requisitos funcionais, requisitos de qualidade e restrições) que são conhecidos neste momento e que estão relacionados a este produto de trabalho.
- *Modificável*: o produto de trabalho é montado de tal forma que pode ser modificado sem degradar sua qualidade.
- *Rastreável*: os requisitos no produto de trabalho podem ser rastreados para trás até suas origens, para frente até sua implementação (em projeto, código e teste), e com outros requisitos dos quais dependem.
- *Conformidade*: se houver instruções obrigatórias de estruturação ou formatação, o produto de trabalho deve estar em conformidade com elas.

3.9 Leitura adicional

Mavin et al. [MWHN2009] introduz e descreve o modelo EARS. Robertson e Robertson [RoRo2012] descrevem os modelos Volere. Goetz e Rupp [GoRu2003], [Rupp2014] discutem

regras e armadilhas para requisitos escritos em linguagem natural. Cockburn [Cock2001] escreveu um livro inteiro sobre como escrever casos de uso. Lauesen [Laue2002] discute descrições de tarefas e também fornece alguns exemplos de produtos de trabalho da ER do mundo real.

A norma ISO/IEC/IEEE 29148 [ISO29148] fornece muitos recursos relativos a produtos de trabalho da ER: templates de frases, critérios de qualidade para requisitos e descrições detalhadas do conteúdo de vários produtos de trabalho da ER, incluindo um modelo de documento para cada produto de trabalho. Cohn [Cohn2010] tem um capítulo sobre como enquadrar os requisitos em um backlog de produto.

Gregory [Greg2016] e Glinz [Glin2016] discutem o problema de como os requisitos detalhados devem ser especificados e até que ponto especificações de requisitos completos e inequívocos são possíveis.

Numerosas publicações tratam da utilização de modelos para especificar os requisitos. A especificação UML [OMG2017], assim como os livros didáticos sobre UML, descrevem os modelos disponíveis em UML. Hofer e Schwentner [HoSch2020] introduzem a modelagem de domínio via Domain Storytelling. [OMG2013] e [OMG2018] descrevem as linguagens de modelagem BPMN para modelagem de processos de negócio e SysML para modelagem de sistemas, respectivamente. Os livros de Booch, Rumbaugh, e Jacobson [BoRJ2005], [JaSB2011], [RuJB2004] dão mais profundidade e aplicações (práticas) da UML. Além disso, os seguintes livros e artigos são recomendados para um conhecimento mais profundo de boas práticas na modelagem de requisitos: [DaTW2012], [Davi1993], [Fowl1996], [GHJV1994]. [LiSS1994] e [Pohl2010] proporcionam uma melhor compreensão dos aspectos de qualidade dos modelos.

4 Práticas para Elaboração de Requisitos

Nos capítulos anteriores, aprendemos sobre a natureza dos requisitos como a representação dos desejos e necessidades das pessoas e organizações para algo novo (por exemplo, um sistema a ser desenvolvido ou adaptado), sobre os princípios subjacentes à produção dos requisitos, e sobre formas de documentar os requisitos. Estabelecemos requisitos antes de construir ou modificar um sistema (ou parte de um) para assegurar que ele seja útil – e aceito – pelas pessoas ou organização que o solicitou. Estes requisitos servem então como insumo para uma equipe de desenvolvimento que irá construir e implementar o sistema.

Em poucas palavras isto é Engenharia de Requisitos; que acontece, explícita ou frequentemente implicitamente, quando e onde as pessoas tentem desenvolver algo. Em princípio, a qualidade dos requisitos determina a qualidade da produção do desenvolvimento subsequente. Sem requisitos adequados, é improvável que o sistema resultante seja útil. Portanto, é importante elaborar os requisitos de forma profissional. Isto requer uma definição explícita de *como*: as práticas a serem utilizadas para uma elaboração de alta qualidade.

É sobre isso que trata este capítulo: ele dá uma visão geral das tarefas, atividades e práticas que são relevantes para qualquer pessoa envolvida na Engenharia de Requisitos. Ela começa com a busca de fontes potenciais de requisitos e termina com a entrega de um conjunto único, consistente, compreensível e acordado de requisitos que podem servir como insumo para o desenvolvimento, manutenção e operação eficiente de um sistema eficaz.

A primeira tarefa em cada esforço de Engenharia de Requisitos será identificar e analisar potenciais *fontes de requisitos*. Isto pode parecer uma tarefa simples e óbvia, mas como veremos na Seção 4.1, há vários aspectos que precisam ser considerados e analisados. A omissão de uma fonte levará inevitavelmente a requisitos pobres ou mesmo inexistentes e, portanto, degradará a qualidade do sistema resultante.

O próximo passo é extrair os requisitos dessas fontes. É como tirar água de um poço: nunca se sabe o que está no balde até que o tenha trazido à superfície. Em Engenharia de Requisitos, essa tarefa é chamada de *elicitação*; é explicada na Seção 0. Na elicitação, transformamos desejos, vontades, exigências, expectativas, o que mais houver, em requisitos explícitos que possam ser reconhecidos e compreendidos por todos os stakeholders.

Entretanto, quando você pergunta a duas pessoas sobre seus requisitos para um determinado sistema, raramente obterá exatamente as mesmas respostas. Em toda uma série de requisitos elicitados de diferentes fontes, é quase certo que algumas delas serão conflitantes. Como é impossível implementar requisitos conflitantes em um mesmo sistema, a *resolução de conflitos* será sempre uma tarefa importante na Engenharia de Requisitos, conforme descrito na Seção 4.3.

A Seção 4.4 é dedicada à tarefa final na Engenharia de Requisitos: a *validação dos requisitos*. O objetivo desta etapa é assegurar que a qualidade do conjunto de requisitos elicitadas e os requisitos individuais dentro deste conjunto sejam suficientemente bons para permitir o desenvolvimento subsequente do sistema.

Da descrição acima das tarefas de Engenharia de Requisitos, você pode ter a impressão de que elas são realizadas como um processo sequencial com uma sequência rigorosa de etapas. No entanto, esta não é certamente a intenção desta descrição e raramente é o caso na prática.

Figura 4.1 mostra algumas etapas do processo que são comuns na Engenharia de Requisitos. Elas podem ser realizadas em paralelo, em loops ou sequencialmente – o que for adequado em cada situação.

O ponto de partida é muitas vezes um conjunto limitado de fontes óbvias. Durante a elicitação destas fontes, novas fontes são identificadas, desencadeando tarefas adicionais de elicitação. Quando conflitos são encontrados, uma elicitação mais detalhada pode ser necessária para encontrar uma saída. Na validação, pode se constatar que uma fonte foi esquecida, um requisito está com defeitos ou um conflito ainda não foi resolvido, resultando em uma nova série de atividades de análise de fontes, elicitação, resolução de conflitos e validação de requisitos. Mesmo durante o desenvolvimento subsequente do sistema, as circunstâncias podem exigir tarefas adicionais de Engenharia de Requisitos.

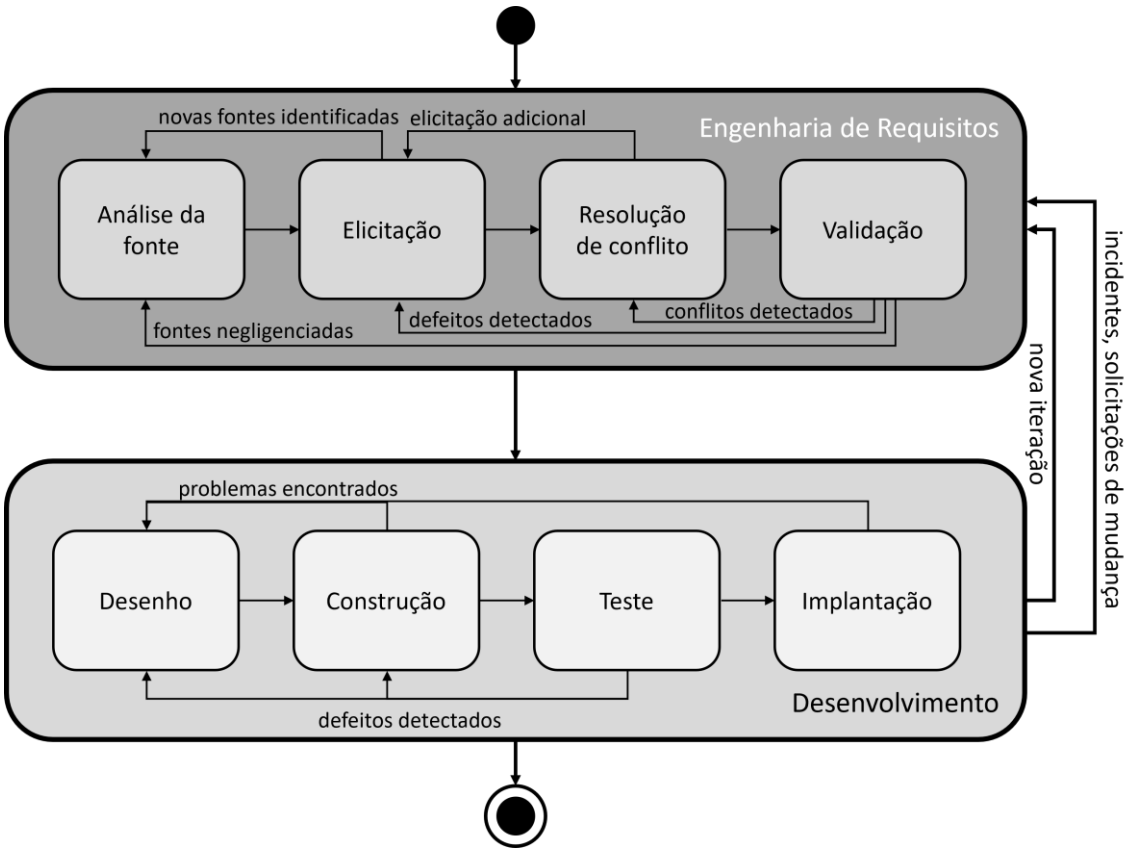


Figura 4.1 Engenharia de Requisitos não é um processo sequencial

Em projetos ágeis, a Engenharia de Requisitos iterativa e incremental e o desenvolvimento de sistemas andam de mãos dadas, com requisitos sendo elaborados pouco antes do desenvolvimento de um novo incremento de sistema. Em tais projetos, você verá com frequência que um projeto começa com um backlog limitado de produtos com requisitos de alto nível que são refinados e detalhados apenas quando são candidatos para a próxima iteração.

4.1 Fontes de Requisitos

Os requisitos não são como barras de chocolate, que ficam na prateleira para que todos possam pegá-las quando e como quiserem. Na introdução deste capítulo, comparamos os requisitos com água a tirada de um poço: é um grande esforço trazê-la para a superfície. Portanto, o primeiro problema que um Engenheiro de Requisitos enfrentará é "Onde estão os poços? Como nenhum requisito vem sem uma fonte, uma das primeiras atividades na elicitação de requisitos é identificar as fontes potenciais. Não basta identificar estas fontes apenas no início de um projeto ou desenvolvimento de produto; isto é um processo que será repetido indefinidamente.

Desde o início da elaboração dos requisitos, o Engenheiro de Requisitos deve estar engajado na identificação, análise e envolvimento de todas as fontes de requisitos relevantes, pois a falta de uma fonte relevante inevitavelmente levará a uma compreensão incompleta dos requisitos relevantes. E isto continuará até o fim: a identificação das fontes de requisitos é um processo que requer constante reavaliação.

O Capítulo 2, Princípio 3 enfatiza a necessidade do *entendimento compartilhado* (explícito e implícito) entre todas as partes envolvidas: stakeholders, Engenheiros de Requisitos, Desenvolvedores. Um entendimento comum do contexto do sistema a ser desenvolvido em um domínio de aplicação específico é um pré-requisito para identificar as fontes relevantes de requisitos. Conhecimento de domínio, colaboração prévia bem-sucedida, cultura e valores comuns e confiança mútua são facilitadores para o entendimento compartilhado, enquanto a distância geográfica, terceirização ou grandes equipes com alta rotatividade são obstáculos.

No Capítulo 2, Princípio 4, introduzimos o contexto como um conceito essencial para compreender e especificar um sistema e seus requisitos. Definimos o contexto como aquela parte da realidade que se situa entre o *limite do sistema* e o *limite do contexto*. As entidades neste contexto influenciarão de alguma forma o sistema ou mesmo interagirão com ele, mas não estarão contidas no próprio sistema.

Isso tornaria a busca por fontes de requisitos bastante simples: basta olhar ao redor no contexto! Mas não é tão fácil assim. No início de um processo de desenvolvimento, o contexto ainda não foi definido; o limite do sistema e o limite do contexto ainda têm que ser determinados. Portanto, a busca de fontes de requisitos é um processo iterativo e recursivo.

As fontes potenciais são analisadas quanto a sua relação com o futuro sistema. Se você não encontrar nenhuma relação ao analisar uma fonte potencial, isto significa que ela faz parte do ambiente irrelevante e não será analisada em relação aos requisitos. Fontes potenciais

que parecem fazer parte do futuro sistema também não têm interesse para o Engenheiro de Requisitos; elas *pertencem* aos desenvolvedores. Somente as entidades com as quais a análise revela uma interação, interface ou influência no futuro sistema, mas que permanecem (relativamente) inalteradas durante o próximo desenvolvimento, merecem atenção como fontes para os requisitos. Nesta análise, o limite do sistema e o limite do contexto são delineados, vagos no início e se tornando mais nítidos à medida que mais e mais fontes são identificadas. À medida que o contexto se torna mais claro, torna-se mais fácil identificar novas fontes, o que, por sua vez, os torna mais precisos.

A busca por fontes de requisitos geralmente começa com algumas fontes óbvias, frequentemente identificadas pelo cliente no início de um esforço de desenvolvimento. A elicitação inicial a partir destas fontes revelará outras fontes potenciais, que são então analisadas para decidir se são ou não relevantes para o sistema. Durante esta análise, novas fontes potenciais podem surgir. Na verdade, em todo esforço de elicitação, o Engenheiro de Requisitos continuará interessado em detectar novas fontes. Isto pode continuar até o final do esforço de desenvolvimento. No entanto, tentamos precocemente identificar as principais e mais relevantes fontes, porque todas as outras atividades de Engenharia de Requisitos dependem dessa identificação precoce.

Na Engenharia de Requisitos, discernimos três categorias principais de fontes:

- Stakeholders
- Documentos
- Sistemas (outros)

Estas categorias são discutidas com mais detalhes nas seções seguintes.

4.1.1 Stakeholders

No Capítulo 2, Princípio 2, você aprendeu que stakeholders são pessoas ou organizações que *influenciam* os requisitos de um sistema ou *são impactadas* por esse sistema.

Os stakeholders de um sistema são as principais fontes para os requisitos. Mais do que ocorre com outras fontes, a não inclusão de um stakeholder relevante terá um importante impacto negativo na qualidade do conjunto final de requisitos; descobrir tais stakeholders tardiamente (ou falhar em identificá-los) pode levar a mudanças caras ou, no final, a um sistema inútil. Para criar um sistema que atenda às necessidades de todos os stakeholders, a identificação sistemática dos stakeholders deve começar no início de qualquer esforço de desenvolvimento e os resultados devem ser gerenciados durante todo o desenvolvimento. Os stakeholders podem ser encontrados em uma ampla área ao redor do sistema, desde usuários diretos e indiretos do sistema, gerentes (de negócio), pessoal de TI como Desenvolvedores e operadores, até adversários e concorrentes, instituições governamentais e regulatórias, e muitos outros. A principal questão para identificar uma pessoa ou uma organização como stakeholder é: "Existe uma relação relevante entre a pessoa/organização e o sistema?

Ajuda a ver os stakeholders como seres humanos feitos de carne e osso. Se você identificar uma organização como stakeholder, faça a si mesmo as seguintes perguntas: "Consigo identificar uma pessoa responsável por esta organização? Quem pode ser visto como o contato principal desta organização? Quem representa esta organização dentro de nossa empresa"? Por exemplo, se o governo é o stakeholder porque uma determinada lei está envolvida, procure alguém que represente o governo como a fonte a ser abordada para os requisitos. Neste caso, não é muito útil identificar o Presidente como essa pessoa; o chefe do departamento jurídico interno seria uma escolha melhor.

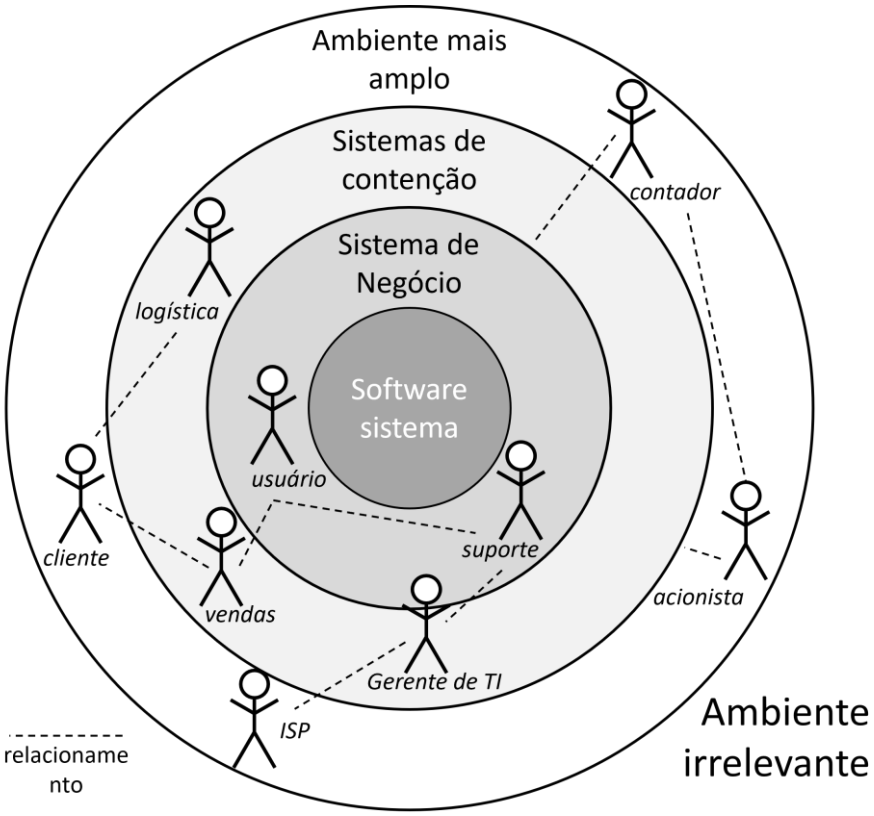


Figura 4.2 O modelo de camadas de Alexander

Não há uma técnica padrão para identificar os stakeholders, mas o modelo de camadas de Ian Alexander [Alex2005] pode ser um bom começo, veja Figura 4.2. Este modelo mostra como um sistema (software) está cercado por várias camadas de sistemas sociotécnicos de alto nível[1] e sistemas sociais, cada um com seus próprios stakeholders. No início de um esforço de desenvolvimento de requisitos, alguns desses stakeholders serão evidentes – por exemplo, usuário ou clientes. Eles podem ser usados como ponto de partida na busca de outros stakeholders. Após identificá-los como fontes relevantes, o Engenheiro de Requisitos analisará suas relações, tanto nos sistemas vizinhos internos como externos. Nesta análise,

[1] Um sistema sociotécnico é um sistema que considera requisitos que abrangem hardware, software, aspectos pessoais e comunitários, ao mesmo tempo em que reconhece a interação entre as complexas infraestruturas da sociedade e o comportamento humano.

novos stakeholders serão encontrados, que por sua vez poderão ter outras (e/ou adicionais) relações a serem analisadas. Você poderia chamar isto de princípio de *bola de neve*: quanto mais stakeholders você encontrar, mais fácil se tornará encontrar novos. Entretanto, ao chegar aos stakeholders no *ambiente mais amplo de* Alexander, quaisquer relações externas acabarão no ambiente irrelevante, o que significa que elas não mais revelarão novas fontes.

Além dos stakeholders indicando outros stakeholders, documentos podem muitas vezes revelar novos stakeholders. Bons exemplos são organogramas, descrições de processos, relatórios de marketing e documentos regulatórios. Para mais informações sobre documentação como fonte de requisitos, consulte a Seção 4.1.2. Checklist de grupos e papéis típicos de stakeholders podem ser uma ferramenta útil para evitar ignorar potenciais stakeholders despercebidos. Além disso, a análise dos stakeholders de sistemas legados ou similares pode ajudar.

Como Engenheiro de Requisitos, você coletará muitos dados sobre seus stakeholders e manterá esses dados até que seu trabalho esteja concluído. Você deve saber quem são os stakeholders, como alcançá-los, quando e onde eles estarão disponíveis, quais são suas especialidades, bem como sua relevância como fonte, quais suas atitudes em relação ao projeto e suas influências sobre ele, quais são seus papéis na empresa, no projeto and na relação com o sistema, etc.

Normalmente, estas informações são mantidas em uma lista de stakeholders e devem ser mantidas atualizadas, pois durante todas as etapas, você permanecerá em contato com todos os stakeholders – algumas de intensa e muito próxima, outras pouco frequentes e superficialmente. Veja Tabela 4.1 abaixo para um exemplo simplificado.

Tabela 4.1 Exemplo de uma lista de stakeholders

Nome	Depto	Telefone	Disponibilidade	Papel	Influencia	Interesse
Marlene	Proprietária	482263	Apenas às segundas-feiras	Patrocinador	++	o
Peter	Vendas	481225	Permanente	Dono do Produto	++	+
Eva	Jurídico	481237	Não em junho	Consultor	+	-
Hassan	Logística	242651	Permanente	Usuário	o	++
Mira	Help desk	242424	Depois das 16h	Usuário	-	+
*Natalia**		481226	Permanente	Cliente	++	++

* Persona, criada, mantida e representada pela equipe *do painel do cliente*

Manter um relacionamento bom e aberto com os stakeholders é fundamental para deles obter informações relevantes. Isto depende principalmente de características comportamentais, tais como integridade, honestidade e respeito.

Comunicação aberta e frequente sobre seus planos, suas atividades e seus resultados é essencial. Você pode ter que transformar os stakeholders de opositores iniciais em colaboradores. Como Engenheiro de Requisitos, você deve entender o que os stakeholders esperam de você. Você também deve *vender* seu trabalho mostrando-lhes os benefícios de sua solução e removendo os obstáculos que os stakeholders experimentam ou esperam no caminho para essa solução. Infelizmente, não é incomum que certos stakeholders (em sua maioria internos) prevejam ou de fato experimentem consequências negativas das mudanças que resultam de seu projeto. Nesses casos, será difícil obter sua cooperação, mesmo que você certamente precise dela. Escalar para níveis superiores na organização pode então ser a única maneira de proceder, mesmo que a relação resultante esteja longe de ser a ideal. O gerenciamento do relacionamento com os stakeholders [Bour2009] é uma forma eficaz de combater problemas com eles.

Isto implica em um processo contínuo de obter e manter o apoio e o compromisso dos stakeholders, engajando os stakeholders certos no momento certo e compreendendo e gerenciando suas expectativas.

A fim de envolver os stakeholders no processo de elicitação, é preciso garantir que eles saibam do que se trata o projeto e qual é seu papel dentro do projeto. Você tem que compreender suas necessidades e tentar atender a essas necessidades na medida do possível dentro de suas competências no projeto. Embora os stakeholders mereçam ser tratados com respeito, você deve exigir o mesmo dos stakeholders, pelo menos daqueles que estão ativamente engajados no projeto. Isto significa que eles devem ter tempo para você quando você precisar deles. Eles devem lhe dar as informações que você pede, assim como outras informações que eles sabem ser relevantes. O feedback deles sobre seus produtos de trabalho deve ser dado em tempo hábil e eles devem se abster de fofocar sobre o projeto etc.

Os problemas com os stakeholders normalmente surgem se os direitos e obrigações do Engenheiro de Requisitos e dos stakeholders com relação ao sistema proposto ou ao projeto atual não forem claros ou se as respectivas necessidades não forem suficientemente atendidas. Se forem encontrados problemas, um tipo de *acordo* ou *contrato* de *stakeholders* pode ajudar a proporcionar à todos a clareza desejada. Quando isto ocorre dentro de uma organização, o endosso da alta administração pode contribuir para o sucesso de tal abordagem.

4.1.1.1 Uma stakeholder Especial: O Usuário

Cada sistema que desenvolvermos terá alguma interação com certos usuários; por que outro motivo você o desenvolveria? Naturalmente, quando você estiver trabalhando nos requisitos de um subsistema técnico embarcado, escondido dentro de algum tipo de maquinário complicado e os usuários só irão interagir com ele indiretamente através de várias camadas de sistemas circundantes. Nesses casos, esses usuários não serão suas fontes mais importantes de Requisitos. Contudo, em muitos sistemas, seres humanos específicos terão uma interface direta com o sistema: os usuários (finais). Seu aceite do

sistema é vital para o sucesso do projeto, portanto eles são seu principal interesse e receberão especial atenção durante toda a Engenharia de Requisitos.

Há duas grandes categorias de usuários:

- *Os Usuários Internos* estão diretamente relacionados com a organização para a qual o sistema está sendo desenvolvido, como funcionários, direção e terceirizados. Costumam ser poucos em número, mais ou menos conhecidos individualmente e de alguma forma envolvidos no projeto. É relativamente fácil contatá-los e eles podem ser alcançados, influenciados e motivados através dos canais formais existentes.
- *Os Usuários Externos* estão fora da empresa, tais como clientes, parceiros de negócio, civis. Seu número pode ser (muito) grande, em muitos casos não são conhecidos individualmente, e podem estar completamente alheios ou indiferentes ao projeto. Você não pode influenciá-los através de canais formais. Para entrar em contato com eles, você pode precisar fazer coisas especiais para motivá-los a participar, como publicidade, prometer alguma recompensa ou dar-lhes acesso gratuito a uma versão beta. Formar um painel de usuários ou dirigir-se ao público (às vezes com pagamento) podem ser formas úteis de envolver esses usuários.

Esteja ciente de que, além destas categorias regulares, também pode ser relevante prestar atenção aos *Usuários Maliciosos*: pessoas que intencionalmente tentam interagir com o sistema de forma prejudicial, tais como hackers ou concorrentes. Raramente é possível contatá-los ou influenciá-los, mas você pode tentar desenvolver medidas para desencorajá-los, mantê-los afastados ou detectar tentativas previsíveis de uso malicioso.

Esta categorização não deve ser considerada de forma muito rigorosa. Podemos imaginar projetos nos quais os usuários externos são em número pequeno e podem ser alcançados facilmente, para que possam ser vistos como *internos*. Por outro lado, nas grandes empresas, a distância até os usuários pode ser tão grande que eles devem ser tratados mais ou menos como *externos*.

Se você tem um bom insight de sua base de usuários, você deve fazer uma distinção entre os papéis destes usuários. Papéis separados geralmente terão necessidades diferentes de informação, utilizarão o sistema à sua maneira e poderão ter direitos de acesso distintos a funcionalidade e dados, por exemplo, usuários que inserem dados versus supervisores que verificam essa entrada. Nesses casos, certifique-se de incluir representantes de todos os papéis relevantes na elicitação.

Muitas vezes, especialmente no início dos esforços de elicitação, esse insight estará ausente. Então, é ainda mais importante perceber que não existe *O Usuário*. *O Usuário* não é uma massa amorfa de seres humanos idênticos, mas uma coleção de indivíduos, cada um com seus próprios hábitos, desejos e necessidades. Quando um sistema tem milhares de usuários ou mais, é claro que você não será capaz de ajustar detalhadamente os requisitos às suas necessidades individuais. Por outro lado, uma abordagem tipo *tamanho único que se encaixa em todos os casos*, visando o usuário *típico* também não funcionará.

Nesses casos, ajuda a discernir vários tipos de usuários ou grupos de usuários que mostram certas semelhanças, muitas vezes comportamentais, dentro do grupo e distintas de outros

grupos. Na prática, ter cerca de três a sete grupos de usuários geralmente funciona melhor. Então, como Engenheiro de Requisitos, você tratará cada grupo como uma fonte distinta de requisitos. Uma boa técnica é o uso de *personas* [Hump2017]. As personas são indivíduos fictícios que descrevem grupos típicos de usuários do sistema com necessidades, objetivos, comportamentos ou atitudes semelhantes.

4.1.1.2 Personas

Há duas abordagens principais para criar personas:

- Baseada em Dados

  Nesta abordagem, personas são desenvolvidas com técnicas de marketing, tais como entrevistas, grupos focais e outras técnicas de coleta de dados etnográficos. Tais personas são chamadas de *personas qualitativas*. Se as personas forem desenvolvidas por meio de análise estatística de uma grande amostra da sua base de usuários, elas serão chamadas de *personas quantitativas*.

- Imaginação

  Como alternativa mais barata e rápida, as personas podem ser imaginadas por exemplo, em uma sessão de brainstorming com a equipe do projeto. Nós as chamamos de *personas ad hoc* ou *proto-personas*. Como Engenheiro de Requisitos, você deve estar ciente de que as personas ad hoc são imaginadas e baseadas em suposições. Essas suposições devem ser verificadas e confirmadas durante todo o processo de Engenharia de Requisitos.

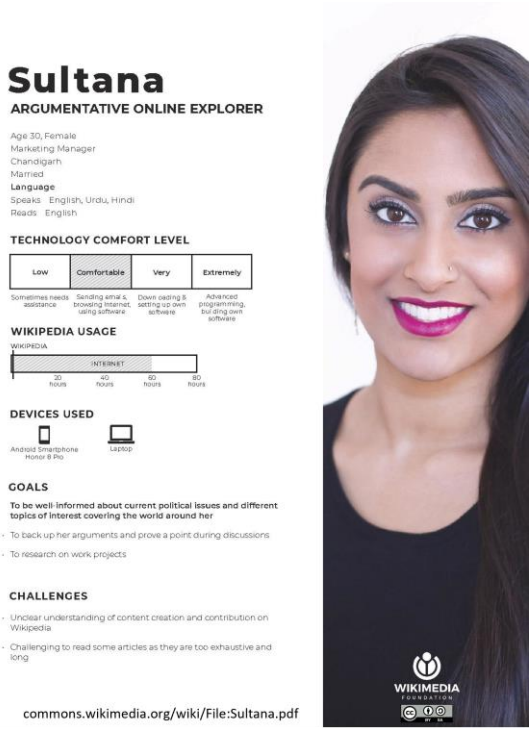


Figura 4.3 Exemplo de Persona

Basicamente, a descrição de uma persona contém informações relevantes para o desenvolvimento do sistema em questão. Normalmente, estas informações serão *enriquecidas* com dados adicionais, tais como nome, endereço, hobbies, e um desenho ou retrato.

Eles dão um rosto humano ao conceito abstrato de persona. Isto pode ajudá-lo a entender que seu trabalho como Engenheiro de Requisitos não se relaciona apenas com fatos, mas também com emoções. Figura 4.3 dá um exemplo da descrição de uma persona.

Se você usar personas no seu projeto, pode ser útil procurar alguns indivíduos que se enquadrem na sua descrições e tratá-los como representantes de cada grupo. Assim você terá stakeholders de carne e osso com os quais poderá se comunicar. Entretanto, lembre-se sempre que o grupo que tal stakeholder representa é um grupo artificial que não existe no mundo real, mas apenas no contexto do sistema ou projeto.

4.1.2 Documentos

Os documentos também podem ser uma fonte importante para os requisitos. Como Engenheiro de Requisitos, muitas vezes é preciso ler muito, especialmente no início de um projeto. Todos os tipos de documentos podem ser relevantes: documentos relacionados a empresas, domínios e projetos, descrições de produtos e processos, documentação legal e regulatória etc. Assim como os stakeholders, você pode fazer uma distinção entre documentos *internos* e *externos*. Os documentos internos podem ser fáceis de obter, mas podem ser confidenciais e não podem ser compartilhados sem consentimento explícito.

Muitas vezes, você precisará assinar um contrato de confidencialidade antes de ter acesso a eles. Documentos externos podem ser difíceis de encontrar, mas geralmente são públicos; se não, certifique-se de que você tenha permissão para acessá-los e usá-los.

Os documentos podem ser uma ótima maneira de encontrar outras fontes. Por exemplo, a descrição de um processo interno pode mencionar certos papéis como estando envolvidos nesse processo, o que, por sua vez, pode levá-lo a novos stakeholders em potencial. Entretanto, os documentos também podem ser fontes diretas de requisitos, especialmente aqueles que são facilmente ignorados ou não regularmente mencionados pelos stakeholders: restrições normativas, diretrizes da empresa e outros documentos legais ou regulatórios; requisitos detalhados de procedimentos e instruções de trabalho; novas ideias brilhantes no material de marketing dos concorrentes. O estudo da documentação pode ajudá-lo a compreender o contexto do sistema a ser desenvolvido, até lendo e-mails entre as pessoas que tomaram a iniciativa do projeto. Ler sobre soluções análogas para problemas e objetivos em outras empresas e domínios pode despertar sua imaginação e mostrar uma direção viável para seu projeto atual.

Como Engenheiro de Requisitos, você deve estar ciente de que um documento está sempre relacionado a algumas pessoas: o(s) autor(es), o público (alvo), um gerente responsável por ele ou um auditor verificando sua aderência, alguém que lhe indicou sua existência etc. Todas essas pessoas podem ter um papel como stakeholder; cabe a você descobrir se este

é ou não o caso. Você deve sempre verificar a validade e a relevância de um documento e muitas vezes precisa que os stakeholders confirmem isso para você. Se você derivasse requisitos de um documento inválido ou desatualizado, o sistema desenvolvido a partir destes requisitos provavelmente falharia.

Assim como os stakeholders, os documentos utilizados como fontes de requisitos devem ser gerenciados. Você pode utilizar uma lista de documentos, comparável à lista de stakeholders discutida acima. Todos os documentos devem ser mantidos em algum tipo de biblioteca comum, indexada, com uma identificação única para permitir que sejam referenciados. Datas e números de versão são importantes para evitar trabalhar com versões desatualizadas; você poderia verificar em intervalos regulares se uma versão mais recente foi publicada e se isso influencia os requisitos. Você deve trabalhar preferencialmente com versões finais, mas na prática, muitas vezes você tem que lidar com versões rascunhos, portanto você também tem que registrar o status dos documentos. Mantenha as versões antigas em um arquivo, porque elas podem ser importantes para entender como um sistema e seus requisitos evoluíram. Gerenciamento adequado e correto dos documentos envolvidos desde o início de um projeto poupará muito trabalho posteriormente na Engenharia de Requisitos, desenvolvimento e implantação. É um bom ponto de partida para estabelecer a rastreabilidade retroativa, conforme discutido na Seção 6.6.

4.1.3 Outros sistemas

Você também pode considerar outros sistemas como fontes para os requisitos do sistema em que está interessado. Aqui, você pode fazer uma distinção entre sistemas *internos* e *externos*, assim como na documentação e com as mesmas considerações sobre acesso e confidencialidade. Outra distinção é a de sistemas *similares* versus sistemas *de interfaces*.

Sistemas similares têm certas funcionalidades em comum com o sistema a ser desenvolvido. Podem ser sistemas predecessores ou legados, sistemas concorrentes, sistemas comparáveis utilizados em outras organizações etc. Muitas vezes você os estuda através de sua documentação, mas às vezes você pode observá-los em ação ou experimentá-los como se eles fossem uma espécie de protótipo. Você poderá entrar em contato com seus usuários para saber mais sobre as funcionalidades e soluções de tais sistemas. Os sistemas predecessores ou legados são frequentemente uma boa fonte para identificar requisitos (funcionais e de qualidade) e restrições detalhados.

Entretanto, esteja ciente de que as restrições (especialmente técnicas) do passado podem não ser mais relevantes e podem não mais restringir seu espaço de solução atual.

Às vezes, um novo sistema e um sistema antigo coexistirão durante um certo período, levando a requisitos adicionais – por exemplo, no que diz respeito ao compartilhamento de dados. Sistemas concorrentes e comparáveis podem ser estudados por suas características de solução e podem ser uma boa fonte de identificação de fatores *inesperados* (ver Seção 4.2.1).

Os sistemas de interfaces terão uma relação direta com o sistema para o qual você está desenvolvendo os requisitos. Eles trocarão dados com seu sistema como fonte e/ou coletor através de alguma interface (síncrona ou assíncrona, em tempo real ou em lote) (veja também a Seção 3.4.2 sobre interfaces de sistema em modelagem de contexto). A configuração, conteúdo e comportamento corretos de tais interfaces são frequentemente essenciais para garantir que seu sistema funcione e, portanto, você terá que entender o sistema em detalhes. Você pode estudar sistemas de interface através de sua documentação, mas como cada detalhe (técnico) é importante aqui, simulação ou testes podem ser necessários. Com relação a sistemas mais antigos ou legados em particular, você não pode confiar que sua documentação esteja atualizada, portanto, você precisará de alguma confirmação disto. Para entender uma interface, você também terá que entender o contexto, a funcionalidade e o comportamento do sistema com o qual a interface se comunica. Será útil se você puder contatar gerentes de aplicações, arquitetos ou designers de tais sistemas, especialmente se o sistema com o qual a interface se comunica tiver que ser atualizado para permitir a nova interface. Esteja também ciente de que um sistema com a qual a interface se comunica terá usuários; pode ser interessante considerar esses usuários como stakeholders no *ambiente social mais amplo* de Alexander de seu próprio sistema.

4.2 Elicitação de Requisitos

Se continuarmos a analogia da água sendo retirada de um poço, estamos agora no ponto em que encontramos o poço e começamos a puxar a corda para encher o balde com a água necessária (ou, neste caso, os requisitos) até a superfície. Isso é o que chamamos de *elicitação*: o esforço gasto pelo Engenheiro de Requisitos para transformar desejos implícitos, demandas, desejos, necessidades, expectativas – que até agora estavam ocultos em suas fontes – em requisitos explícitos, compreensíveis, reconhecíveis e verificáveis. Naturalmente, para ficar completo, teremos que retirar água de todos os poços e puxar a corda da maneira correta para garantir que toda a água chegue à superfície. Na terminologia da Engenharia de Requisitos, dizemos que devemos aplicar as *técnicas de elicitação* corretas.

Uma categorização comum das técnicas de elicitação é a distinção entre:

- Técnicas de coleta
- Técnicas de Desenho e Geração de Ideias

A partir destas categorias, você pode selecionar uma ampla gama de técnicas de elicitação, cada uma com suas próprias características. Figura 4.4 dá uma visão geral das técnicas de elicitação em suas categorias e subcategorias.

Figura 4.4 Uma Visão Geral das Técnicas de Elicitação

Uma competência essencial do Engenheiro de Requisitos é a capacidade de escolher as (combinações de) técnicas corretas para cada circunstância. A escolha das corretas depende de muitos fatores, como por exemplo:

- Tipo de sistema

  Um novo sistema completamente inovador se beneficiará mais das técnicas de criatividade, enquanto um sistema de substituição em um ambiente critico de segurança precisará de técnicas de questionamento e arqueologia do sistema.

- Modelo de ciclo de vida de desenvolvimento de software

  Em um projeto em cascata, você pode ter planejado utilizar técnicas que exigem mais tempo como aprendizagem ou analogias, enquanto em um ambiente ágil, podem prevalecer brainstorming, storyboarding e prototipação.

- Pessoas envolvidas

  Por exemplo, a observação de campo provavelmente não será apreciada em negócios altamente confidenciais; uma pesquisa abrangente pode ser preferida em vez de um grande número de entrevistas individuais.

- Configuração organizacional

  Uma organização governamental estabelecida precisa de uma abordagem totalmente diferente de uma startup; uma empresa distribuída e altamente

descentralizada precisa de uma abordagem diferente de uma empresa estabelecida em um único local.

Os melhores resultados são geralmente obtidos com uma combinação de diferentes técnicas de elicitação. Para uma abordagem sistemática para selecioná-los, ver [CaDJ2014].

As técnicas de elicitação são – ou pelo menos, devem ser – capazes de detectar todos os tipos de requisitos. Na prática da Engenharia de Requisitos, porém, os *requisitos funcionais* explícitos são muitas vezes supervalorizados, e os *requisitos de qualidade* e *restrições* mais implícitos recebem menor atenção.

Isto pode resultar em um sistema que – com todos os requisitos funcionais sendo cumpridos – não funciona, tem má usabilidade, não cumpre as diretrizes arquitetônicas, ou não atende a outros requisitos de qualidade ou restrições e, consequentemente, não será aceito.

Os stakeholders podem ser fontes, mas muitas vezes você encontrará mais informações em documentos. Para a elicitação dos *requisitos de qualidade*, a aplicação de uma lista de verificação baseada na norma ISO 25010 [ISO25010] pode ajudar a detectá-los e quantificá-los – por exemplo, na preparação para uma entrevista. As *restrições* podem ser encontradas considerando possíveis limitações do espaço potencial de solução, por exemplo, questões técnicas, arquitetônicas, legais, organizacionais, culturais ou ambientais. A documentação relevante pode muitas vezes ser identificada através de membros da equipe.

4.2.1 O Modelo Kano

Uma das principais circunstâncias a considerar na seleção de uma técnica de elicitação é a natureza e a importância de um requisito que estamos tentando descobrir. Para obter mais informações sobre a natureza de certos requisitos, o modelo Kano [Verd2014] vem a calhar. Este modelo, mostrado na Figura 4.5, classifica características de um sistema em três categorias:

- *Inesperados* (sinônimos: fatores de excitação, requisitos inconscientes)

  Uma funcionalidade inesperada é a desconhecida pelos stakeholders; por isso chamada de *inconsciente*. Os stakeholders não pedem a funcionalidade porque não sabem ser possível no sistema – por exemplo, um smartphone que pode ser transformado em um projetor. No início, quando a funcionalidade é nova no mercado, a maioria dos clientes terá suas dúvidas sobre ela, mas quando alguns dos primeiros adotantes a experimentaram e começaram a compartilhar suas experiências, mais e mais pessoas a quererão. Se um fator inesperado estiver ausente, ninguém reclamará; mas quando presente, este pode ser um diferencial que atrai muitos clientes.

- *Esperados* (sinônimos: fatores de desempenho, requisitos conscientes, satisfiers)

  Um fator esperado é uma funcionalidade que os clientes pedem explicitamente (daí requisitos *conscientes*). Quanto mais fatores esperados você coloca no seu sistema,

maior será a satisfação de seus clientes. Um exemplo poderia ser o número de lentes e opções de vídeo em um smartphone. Como acrescentar funcionalidades esperadas geralmente também significa custos mais altos, muitas vezes será necessária uma espécie de análise de custo/benefício para decidir quantos deles serão incorporados ao sistema.

- *Básicos* (sinônimos: fatores básicos, requisitos subconscientes, dissatisfiers)

  Fatores básicos também são funcionalidades que os clientes não pedem. Aqui, porém, a razão para não o pedir é que o requisito é tão óbvio *(daí subconsciente)* que os clientes não podem imaginar que não faça parte do sistema; estes requisitos são tacitamente considerados indispensáveis. Imagine um smartphone sem GPS. Se um fator básico for incluído como feature de um sistema, os clientes não o notarão porque pensam que o sistema não poderia existir sem ele. No entanto, se você ignorar tal requisito e deixá-lo de fora do sistema, os clientes ficarão muito chateados e se recusarão a usar o sistema.

O modelo Kano olha os requisitos da perspectiva do stakeholder. Ela concentra-se em diferenciar funcionalidades, em oposição às necessidades expressas. Com o modelo Kano em mente, você pode encontrar mais requisitos do que quando se concentra apenas nas necessidades explicitamente formuladas pelos stakeholders. Como veremos mais adiante neste capítulo, todas as categorias KANO podem ser ligadas a distintas técnicas de elicitação.

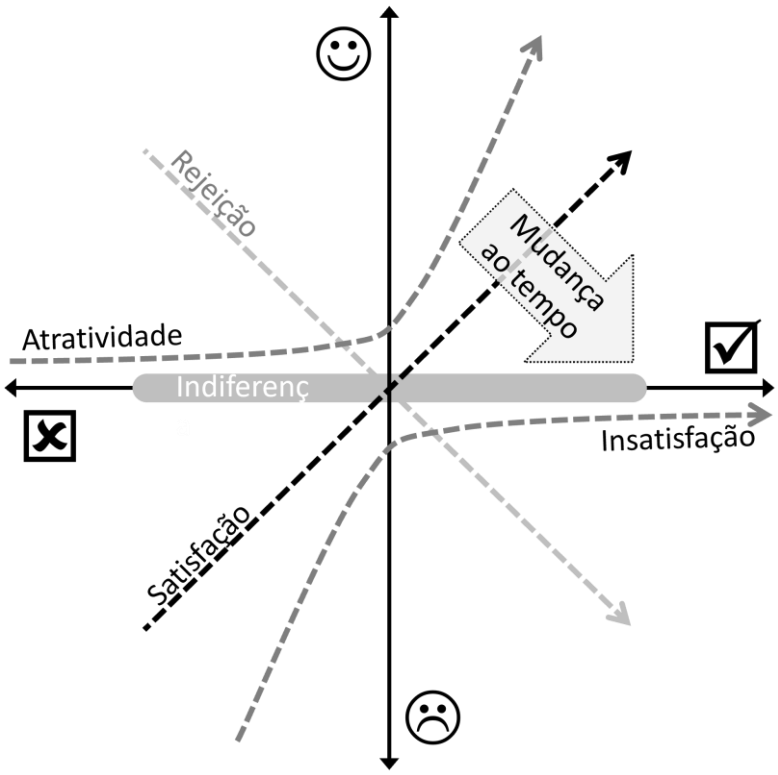


Figura 4.5 O Modelo Kano

Na verdade, o modelo Kano original contém mais duas categorias, a indiferente (ou eu não me importo) e os rejeitados (ou eu odeio). Estas categorias não recebem muita atenção na maioria dos manuais de Engenharia de Requisitos, mas ainda podem ser úteis para você como Engenheiro de Requisitos. Suponha, por exemplo, que os desenvolvedores queiram adicionar uma determinada funcionalidade ao sistema por razões técnicas. Se, após a análise, você verificar que os clientes são indiferentes a esta funcionalidade, é seguro incluí-la no sistema. No entanto, se isso se revelar um requisito de rejeição, você deve dizer aos desenvolvedores para procurarem uma alternativa menos prejudicial, pois a implementação deste requisito pode se revelar um erro dispendioso.

Uma observação interessante quando se trabalha com o modelo Kano é que os requisitos tendem a mudar com o tempo. Se alguém introduz uma nova funcionalidade, não há certeza sobre como o mercado reagirá a ela. Às vezes, os clientes ficarão indiferentes e a funcionalidade só sobreviverá se não aumentar o preço do produto.

Se os clientes a rejeitarem, a funcionalidade provavelmente será removida do produto o mais rápido possível. Entretanto, quando (talvez inicialmente uma vanguarda de) os clientes gostam da funcionalidade , ela se tornará um fator de entusiasmo, um argumento de venda único pelo qual os clientes estão preparados a pagar o preço. À medida que mais e mais clientes descobrirem, experimentarem e apreciarem esta nova funcionalidade , ela se tornará um fator esperado, explicitamente solicitado. Gradualmente, quando sistemas semelhantes começam a implementar a mesma funcionalidade , os clientes podem esquecer que os sistemas não incluíam originalmente tal feature e a tomarão como certa, transformando-a em um fator básico. É por isso que muitos sistemas contêm funcionalidade que os usuários consideram indispensáveis sem saber o porquê e, portanto, sem pedir explicitamente por elas.

Um bom exemplo é a função de câmera em telefones celulares, para a qual este processo levou menos de 20 anos. A primeira vez que uma câmera foi introduzida como parte de um telefone celular, a maioria dos clientes ficou intrigada: ninguém havia solicitado este recurso e a maioria dos clientes pensou "Se eu quiser tirar uma foto, utilizo uma câmera". Entretanto, alguns dos primeiros adotantes experimentaram e descobriram a conveniência de tirar fotos sem uma câmera, poder compartilhá-las instantaneamente com outras pessoas sem precisar revelá-las antes. Eles gostaram da funcionalidade de uma câmera como um fator inesperado e assim todas as marcas começaram a implementá-la em seus telefones, tornando-a em um fator esperado: quanto melhor a qualidade das fotos, mais satisfeitos ficavam os usuários. Hoje em dia, ao comprar um novo celular, todos tomam como certo que ele terá uma função de câmera, de modo que se tornou um fator básico: "Se eu não posso tirar uma foto com este telefone celular, não presta."

Como você categoriza uma funcionalidade específica? Usando a Análise Kano. Para uma funcionalidade específica, você faz duas perguntas a um grupo representativo de potenciais stakeholders: (1) "O que você sentiria se essa feature estivesse presente no sistema"? e (2) "O que você sentiria se essa funcionalidade estivesse ausente do sistema? Você os deixa marcar as respostas numa escala de 5 pontos entre "Eu amo isso" e "Eu odeio isso" e depois marque a resposta média na matriz de Análise Kano, como mostrado em Figura 4.5. A célula

que aparece no cruzamento das duas respostas fornece a classificação Kano para a feature.

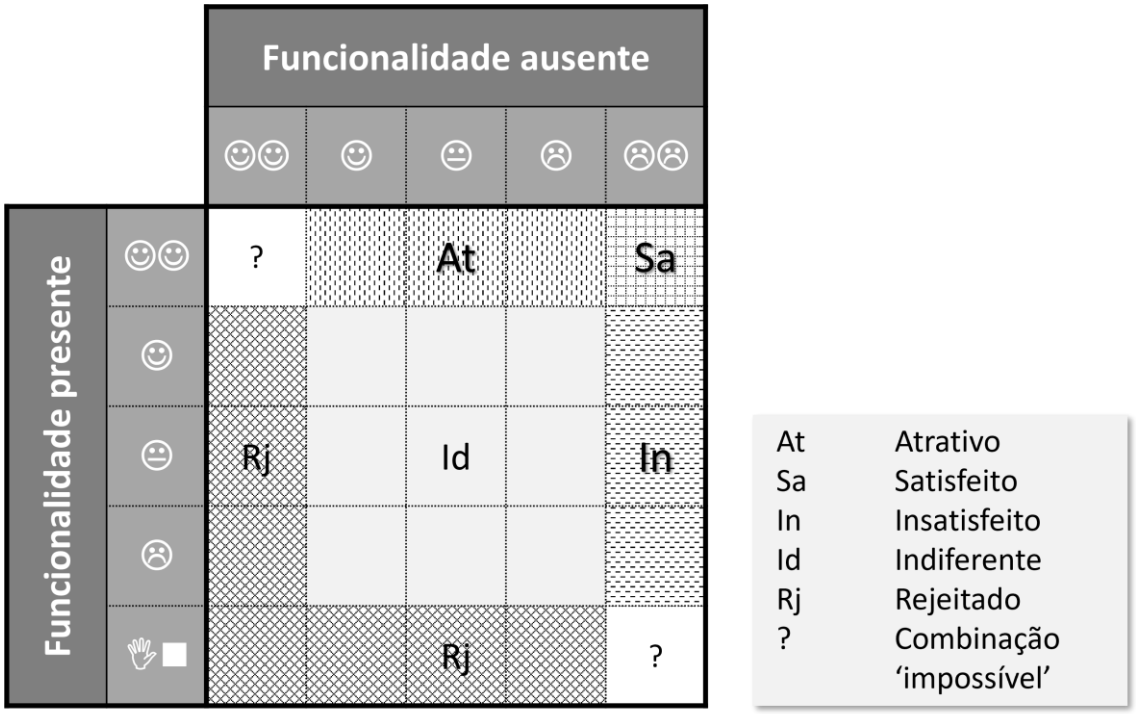


Figura 4.6 Matriz de Análise Kano

A próxima pergunta é: por que se preocupar com a análise Kano na elicitação de requisitos?

Como explicamos nas seções seguintes, você precisará de diferentes técnicas para encontrar estas diferentes categorias de funcionalidades. Por si só, os stakeholders falarão principalmente de fatores esperados – seus requisitos conscientes que eles explicitamente pedem. É muito mais difícil detectar as outras categorias mas, felizmente, existem várias técnicas úteis para fazê-lo.

4.2.2 Técnicas de Coleta

Com as *Técnicas de Coleta,* você examina as diferentes fontes que você identificou e levanta os requisitos a partir daí. Estas técnicas são bem estabelecidas e têm sido comumente usadas em toda a Engenharia de Requisitos e predominantemente produzem fatores esperados e básicos.

As técnicas de coleta podem ser ainda subdivididas em quatro categorias:

- Técnicas de Questionamentos
- Técnicas de Colaboração
- Técnicas de Observação
- Técnicas baseadas em Artefatos

As *técnicas de questionamento* são sempre utilizadas em uma interação com os stakeholders. O Engenheiro de Requisitos faz perguntas apropriadas aos stakeholders a fim

de permitir que eles reflitam e assim receber respostas das quais os requisitos possam ser derivados.

São exemplos de técnicas de questionamento:

- Entrevistas

  Devido a sua flexibilidade, as *entrevistas* são provavelmente uma das técnicas de elicitação mais frequentemente utilizadas. Elas não requerem ferramentas específicas e podem ser usadas para elicitar requisitos tanto abstratos quanto específicos. Normalmente, uma entrevista é uma sessão individual entre um engenheiro de requisitos (entrevistador) e um stakeholder (entrevistado), mas um pequeno grupo de entrevistados pode ser uma opção. Normalmente, os requisitos elicitados em uma entrevista são fatores esperados, já que o entrevistado expressa informações conscientes. A técnica da entrevista não é muito complicada e a maioria das pessoas tem uma boa compreensão do que ela é. Entretanto, você precisa de objetivos claros e boa preparação para obter resultados úteis. As entrevistas podem revelar informações detalhadas e permitem flexibilizar a condução da entrevista com base nas respostas dadas. Elas consomem bastante tempo, portanto, esta técnica é menos apropriada quando se deseja entrevistar um grande número de stakeholders.

- Questionários

  Com um *questionário*, um grupo maior de stakeholders é convidado a responder – oralmente ou por escrito, ou em uma página da Web – o mesmo conjunto de perguntas, que são apresentadas de forma estruturada. Questionários quantitativos são usados para confirmar hipóteses ou requisitos previamente elicitados. Eles utilizam perguntas fechadas (somente respostas pré-definidas permitidas) e podem, portanto, rapidamente ser avaliados e fornecer informações estatísticas. Por outro lado, os questionários qualitativos utilizam perguntas abertas e podem encontrar novos requisitos. Eles tendem a fornecer resultados complexos e, portanto, geralmente são mais demorados para preparar e avaliar. Em geral, os questionários são uma técnica preferida para grandes grupos. Esteja ciente, entretanto, de que a elaboração de um bom questionário envolve um grande esforço. Um questionário é frequentemente o próximo passo após a obtenção de uma ideia preliminar baseada em uma série de entrevistas, a fim de validar estas ideias dentro de um grupo maior.

Na categoria de *técnicas de colaboração,* encontramos todos os tipos de colaboração entre o Engenheiro de Requisitos e outras pessoas (stakeholders, especialistas, usuários, clientes etc.). Alguns exemplos são:

- Workshops

  Workshop é um termo genérico para técnicas orientadas a grupos, desde pequenas reuniões informais até eventos organizados com várias dezenas ou mesmo centenas de stakeholders. Uma boa definição é a seguinte: "Um workshop de requisitos é uma reunião estruturada na qual um grupo cuidadosamente selecionado de stakeholders e especialistas no assunto trabalham juntos para definir, criar, refinar e chegar a um acordo sobre os entregáveis (tais como modelos e documentos) que representam os

requisitos do usuário" [Gott2002]. Com um workshop, você pode obter um bom insight global em pouco tempo, pois utiliza a interação entre os participantes. Se você precisar de mais detalhes, entrevistas complementares ou questionários são apropriados. Os workshops podem servir tanto como técnica (de elicitação) de coleta quanto como técnica de criatividade (ver Seção 4.2.3).

- Engenharia de Requisitos baseada em multidão

  Na Engenharia de Requisitos baseada *em multidões* (também conhecida como baseada em plataformas) (ver [GreA2017]), a elicitação se transforma em um esforço participativo com uma multidão de stakeholders, em particular os usuários, levando a requisitos mais precisos e, em última instância, a um software melhor. O poder da multidão reside na diversidade de talentos e conhecimentos disponíveis. Como a quantidade de dados obtidos da multidão será grande, uma plataforma automatizada para processar esses dados é essencial. Esta plataforma deve oferecer funcionalidades orientadas à multidão que apoiem a colaboração e o compartilhamento de conhecimento e promovam o envolvimento de grupos maiores de stakeholders na coleta, análise e desenvolvimento de requisitos de software, bem como a validação e priorização desses requisitos de forma dinâmica e orientada ao usuário.

As *técnicas de observação* também são aplicadas em relação aos stakeholders. Os stakeholders são observados enquanto estão envolvidas em seus processos normais (de negócio) em seu contexto habitual, sem interferência direta do Engenheiro de Requisitos. As técnicas de observação são particularmente úteis para identificar os fatores básicos. Você pode observar atividades peculiares, sequências, dados etc., que são tão comuns aos stakeholders que eles não as mencionam, e estes aspectos, portanto, não vêm à tona facilmente nas técnicas de coleta.

As formas comuns de técnicas de observação são:

- Observação de campo

  Durante a *observação de campo*, o Engenheiro de Requisitos observa (principalmente) os usuários finais em seu ambiente enquanto esses usuários realizam as atividades para as quais um sistema deve ser desenvolvido. A observação de campo é normalmente usada em situações onde a interação distrairia os usuários ou interferiria no próprio processo e potencialmente falsificaria os resultados. Pode até ser aplicado sem informar os sujeitos observados, por exemplo, sentando-se com outros pacientes na sala de espera de um dentista para observar seu comportamento. Com a observação de campo, você será capaz de detectar requisitos (muitas vezes de forma detalhada) que não seriam facilmente encontradas com outras técnicas – por exemplo, quando ações e comportamentos são muito difíceis de serem descritos verbalmente.
  Esteja ciente de que a observação de campo requer muita preparação, um olhar atento e muito tempo. Gravar um vídeo pode ser bastante útil para capturar o comportamento dos stakeholders. Pode ser usado em conjunto com a observação

direta de campo e pode até substitui-la em situações em que a presença do Engenheiro de Requisitos não é permitida ou desejada. O vídeo oferece a possibilidade de pós-processamento para permitir a investigação detalhada de atos e procedimentos de difícil observação.

- Aprendizagem

  A *Aprendizagem* difere da observação de campo por ser participativa. Na aprendizagem, o Engenheiro de Requisitos *(aprendiz*) faz uma espécie de estágio no ambiente no qual o sistema em questão será usado (ou já está em uso) e usuários experientes *(mestres*) ensinam o aprendiz como as coisas funcionam. O aprendiz participa, mas não interfere; ele age como um novato no campo e podem cometer erros e fazer perguntas "bobas". O objetivo é criar uma compreensão profunda do domínio, do negócio e dos processos antes que se inicie a real elicitação dos requisitos. Muitas vezes será necessário um acompanhamento com entrevistas e questionários para verificar as ideias iniciais. Os requisitos resultantes podem ser documentados e validados posteriormente. Uma duração ótima para tal estágio depende de muitos fatores diferentes (por exemplo, complexidade do processo, repetitividade, disponibilidade de tempo do mestre e aprendiz), mas geralmente varia entre um dia e várias semanas. Esteja ciente de que a aprendizagem pode ser difícil ou impossível de organizar em certos domínios, tais como medicina, aviação ou militares.

As técnicas baseadas em artefatos não utilizam stakeholders (diretamente), mas sim produtos de trabalho como documentos e sistemas, ou mesmo imagens, arquivos de áudio e vídeo, como fontes para os requisitos. Estas técnicas podem encontrar (às vezes detalhadamente) fatores esperados e básicos. Normalmente examinar estes produtos de trabalho é demorado (muitas vezes estão mal estruturados, desatualizados ou parcialmente irrelevantes). No entanto, técnicas baseadas em artefatos podem ser úteis, particularmente quando os stakeholders não estão prontamente disponíveis.

Alguns exemplos de técnicas baseadas em artefatos:

- Arqueologia de sistemas

  Na *arqueologia de sistemas*, os requisitos são extraídos de sistemas existentes – como sistemas legados, sistemas concorrentes, ou mesmo sistemas análogos – analisando sua documentação (design, manuais) ou implementação (código, comentários, scripts, histórias de usuários, casos de teste). Esta técnica é utilizada principalmente se um sistema existente foi usado por muitos anos e agora deve ser substituído por um novo sistema por qualquer razão; o novo sistema tem que cobrir a mesma funcionalidade que o antigo, ou pelo menos certas partes dele. A arqueologia do sistema frequentemente leva muito tempo, mas pode revelar requisitos e restrições detalhadas que não são facilmente detectadas de outra forma. Entretanto, você precisará de tempo extra para verificar, através de outros canais, se estes requisitos ainda são válidos e relevantes.

- Análise de feedback

  Há muitas maneiras de coletar *feedback* de (potenciais) usuários e clientes, seja em um sistema existente ou em um protótipo. Os dados de feedback podem ser estruturados (por exemplo, uma classificação 5 estrelas em uma loja de aplicativos) ou não estruturados (como comentários de revisão). Ele pode ser coletado através de pesquisas na web e contatos via formulários, durante testes beta ou testes A/B, em mídias sociais, ou mesmo como observações de clientes recebidas em um call center. Muitas vezes, a quantidade de dados é bastante grande, e a análise será demorada. No entanto, o feedback pode ser muito útil para obter informações sobre as *dores e ganhos* do usuário. Pontuações negativas e observações críticas o ajudarão a detectar fatores básicos despercebidos. Pontuações positivas e elogios lhe darão informações adicionais sobre fatores esperados. Ocasionalmente, os comentários podem até conter ideias inovadoras que podem ser transformadas em fatores de entusiasmo. A análise de feedback pode assim resultar no ajuste dos requisitos existentes, mas também na descoberta de novos requisitos.

- Reutilização de requisitos

  Muitas organizações já têm uma grande coleção de requisitos que foram levantados e elaborados no passado para sistemas anteriores. Muitos desses requisitos também podem ser aplicados a um novo sistema, especialmente requisitos que foram derivados de um modelo de domínio abrangente. Portanto, a *reutilização* de requisitos existentes pode economizar muito tempo e dinheiro porque você pode pular a sua elicitação. Entretanto, isto só funciona se esta coleção de requisitos existente estiver atualizada, gerenciada de forma eficaz, facilmente disponível e amplamente documentada, o que infelizmente não é o caso com frequência. Mesmo que a reutilização seja viável, esteja ciente de que você ainda precisa validar com os stakeholders se os requisitos reutilizados são relevantes e válidos na nova situação, seja diretamente ou com alguns ajustes.

4.2.3 Técnicas de Desenho e Geração de Ideias

No passado, a Engenharia de Requisitos concentrou-se em reunir e documentar os requisitos necessários de todos os stakeholders relevantes, aplicando técnicas de coleta, conforme introduzido na seção anterior. A crescente influência do software como um impulsionador de inovação em muitas empresas está agora exigindo cada vez mais um novo posicionamento da Engenharia de Requisitos como uma atividade criativa e de resolução de problemas. Isto envolve a aplicação de outras técnicas que não mais consideram os stakeholders (e seus documentos e sistemas) a única fonte de requisitos. Sistemas inovadores precisam de features novas, talvez disruptivas, que os atuais stakeholders não podem imaginar (ainda).

As *técnicas de desenho e geração de ideias* surgiram para atender a esta necessidade. Estas técnicas promovem a criatividade, principalmente dentro de equipes, para a geração de ideias e podem fornecer maneiras adicionais de elaborar uma determinada ideia. Essas

técnicas podem gerar requisitos novos e inovadores que, muitas vezes, encantam. Muitas técnicas diversas existem dentro desta ampla categoria, algumas notavelmente simples, outras bastante elaboradas. Analisaremos alguns exemplos de duas subcategorias:

- Técnicas de Criatividade
- Técnicas de Desenho

Além disso, vamos olhar para o campo emergente do *Design Thinking*.

As técnicas de criatividade estimulam a criatividade para encontrar ou criar novos requisitos que não podem ser coletados diretamente dos stakeholders porque estes não estão cientes da viabilidade de certas novas features ou inovações (técnicas). Estas técnicas são normalmente aplicadas dentro de equipes diversas e multidisciplinares de pessoal de TI, tais como analistas, Engenheiros de Requisitos, desenvolvedores, testadores, Product Owners, gerentes de aplicações etc., com ou sem representantes de negócio, usuários, clientes e outros stakeholders. As técnicas estimulam o pensamento "fora da caixa" e "sem barreiras" bem como a elaboração das ideias uns dos outros. Infelizmente, nenhuma delas garante o sucesso na geração de resultados criativos, pois vários mecanismos em nosso cérebro têm que se unir para possibilitar ideias criativas.

> Um exemplo óbvio onde as técnicas de criatividade são importantes é a indústria de jogos. É claro que você pode perguntar aos jogadores sobre seus requisitos com uma técnica de coleta e aprenderá o que os jogadores gostam ou não gostam nos jogos atuais. Entretanto, para desenvolver um jogo de sucesso, você precisa surpreender os jogadores com algo novo; você tem que descobrir seus fatores de entusiasmo. É exatamente aí que se encaixam as técnicas de criatividade.

Vários pré-requisitos foram identificados como fatores importantes para estimular a criatividade:

- Oportunidade – e, portanto, tempo – de surgir uma ideia
- Conhecimento do assunto, o que eleva a possibilidade de surgir uma ideia que faça a diferença
- Motivação, pois nosso cérebro só pode ser criativo se houver um benefício direto para a pessoa
- Segurança e proteção, pois ideias inúteis não devem ter consequências negativas

Dois exemplos de técnicas de criatividade são apresentados:

- Brainstorming

  Brainstorming (ver [Osbo1948]) apoia o desenvolvimento de novas ideias para uma determinada questão ou problema. Como na maioria das técnicas de criatividade, o ponto crucial do brainstorming é adiar o julgamento de ideias, separando a ideia apresentada da sua análise. Algumas diretrizes gerais para o brainstorming incluem:
  A quantidade (de ideias) prevalece sobre a qualidade.
  A livre associação e o pensamento visionário são explicitamente encorajados e promovidos.

Considerar e combinar ideias apresentadas é permitido e desejado.
É proibido criticar as ideias de outros participantes, mesmo que uma ideia pareça absurda.
Após uma sessão de brainstorming, as ideias que surgiram são categorizadas, avaliadas e priorizadas. As ideias selecionadas servem, então, como insumo para uma elicitação adicional.

- Técnica de analogia

  A *técnica de analogia* (ver [Robe2001]) ajuda no desenvolvimento de ideias para tópicos críticos e complexos. Ela usa analogias para apoiar o pensamento e a geração de ideias. Seu sucesso ou fracasso é influenciado principalmente pela seleção de uma analogia adequada para o problema em questão. A analogia selecionada pode ser próxima (por exemplo, o mesmo problema em outro negócio) ou distante (por exemplo, comparar uma organização com um organismo vivo) do problema original. A aplicação da técnica de analogia consiste em duas etapas:
  Elaborar os aspectos da analogia selecionada em detalhes sem se referir ao problema original.
  Transferir todos os aspectos identificados da analogia de volta ao problema original.
  Os conceitos e ideias resultantes serão então um ponto de partida para uma elicitação adicional.

As técnicas de desenho ajudam a explorar e elaborar ideias geradas com técnicas de criatividade e também ajudam a esclarecer e concretizar necessidades vagas das partes interessadas. Eles dependem muito de artefatos visuais ou tangíveis, da cooperação da equipe e do feedback do cliente.

As técnicas populares nesta categoria incluem:

- Protótipos

  Por *protótipo* (em relação à elicitação; consulte também a Seção 3.7 para mais informações), entendemos um tipo de produto de trabalho intermediário que é criado ou implantado para gerar feedback. Os protótipos podem variar de simples esboços em papel a versões pré-lançamento operacional de um sistema. Eles permitem aos futuros usuários experimentar o sistema de forma mais ou menos tangível e investigar certas características, ainda não claras, durante a Engenharia de Requisitos e antes da implementação real. Como veremos na seção sobre validação (4.4.2), os protótipos são utilizados principalmente para verificar se os requisitos previamente definidos foram implementados corretamente. Entretanto, com a devida orientação dos usuários e análise de seu feedback, esta técnica também pode ser usada para derivar novos requisitos. Pode ser particularmente útil para detectar requisitos de qualidade, fatores básicos de insatisfação e restrições, ou quaisquer outras características que não possam ser facilmente compreendidas ou definidas antecipadamente através de modelos e documentação.

- Cenários e Storyboards

  A palavra *cenário* deriva do teatro, onde é usada para se referir a um esboço de uma peça de teatro, ópera ou similar, indicando uma sequência de cenas com seus personagens. Em TI, usamos este termo para descrever um fluxo de ações para um sistema, incluindo os usuários envolvidos (que normalmente chamamos de atores). Através de cenários, você pode explorar formas alternativas de realizar um processo em um sistema. Devido à sua estrutura simplificada, eles são fáceis de desenvolver e podem ser mudados rapidamente. Da mesma forma que para protótipos, cenários e storyboards podem ser aplicados tanto na elicitação inicial quanto na validação final dos requisitos.
  Os cenários podem ser documentados de forma escrita ou visual. A forma visual de um cenário é chamada de *storyboard*.
  Um storyboard é normalmente uma espécie de história em quadrinhos com uma série de painéis que mostram a interação de certas personas com o sistema. Veja Figura 4.7 para um exemplo. Cenários e storyboards são úteis para a elaboração inicial de ideias em termos de processos e atividades.

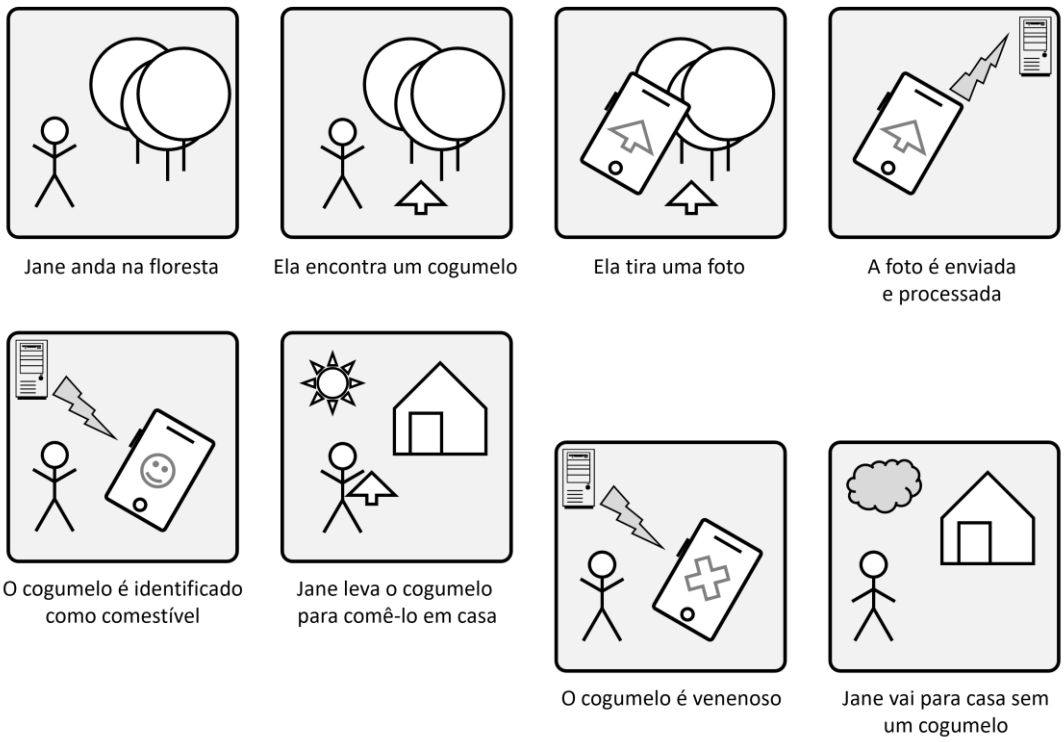


Figura 4.7 Exemplo de um Storyboard

Design Thinking não é tanto uma técnica, mas sim um conceito, uma atitude, uma filosofia, uma família de processos e muitas vezes uma caixa de ferramentas cheia de técnicas. O foco está na inovação e na solução de problemas. Existem várias variantes de Design Thinking, a maioria usando técnicas simples, visuais e ágeis. Dois princípios básicos podem ser encontrados em todas as variantes:

- Empatia

  O primeiro passo para os praticantes de Design Thinking é encontrar o verdadeiro problema por trás de dado problema. Eles tentam entender o que os stakeholders realmente pensam, sentem e fazem quando interagem com um sistema. Portanto, muitas vezes nos referimos ao Design Thinking como design centrado no ser humano. Personas, mapas de empatia e co-criação com clientes são técnicas comuns para este fim.

- Criatividade

  Uma característica comum do Design Thinking é o conceito representado por um *diamante*: a alternância do pensamento divergente e convergente. O pensamento divergente visa explorar uma questão mais ampla e profunda, gerando muitas ideias diferentes, e o pensamento convergente focaliza, seleciona, simplifica e combina essas ideias em uma única entrega final. Um padrão básico, o modelo de *diamante duplo*, é mostrado em Figura 4.8 (ver [DeCo2007]).

Um tratamento detalhado de Design Thinking está além do escopo deste guia de estudo para o Nível Fundamental.

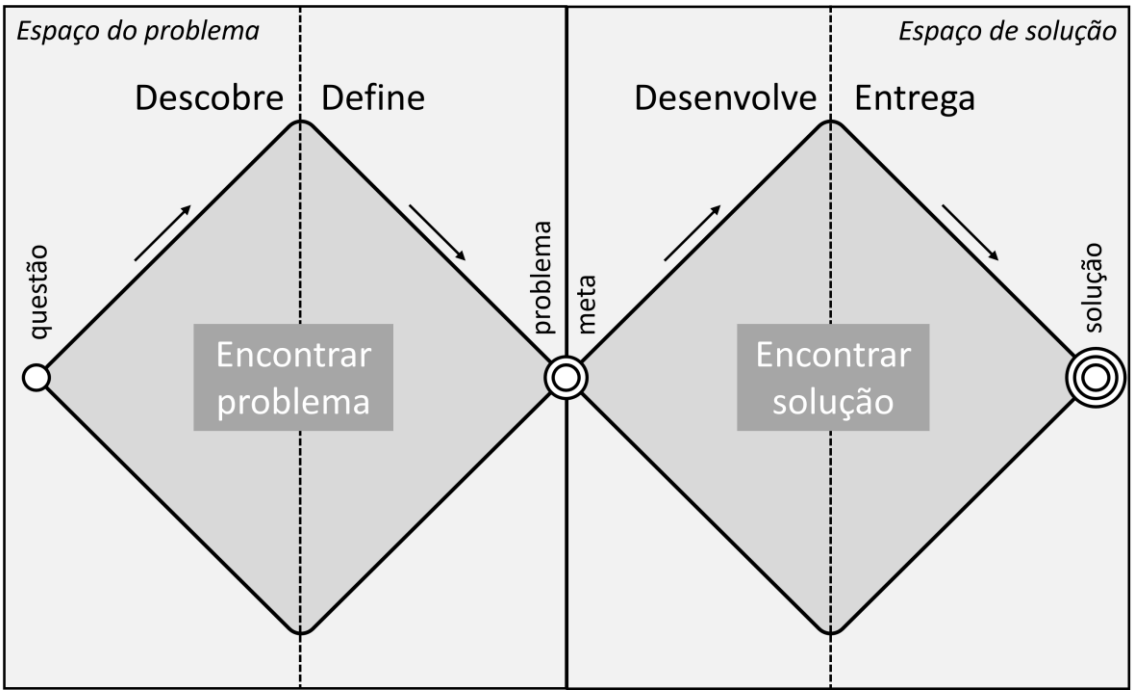


Figura 4.8 O Diamante Duplo

4.3 Resolução de Conflitos em Relação aos Requisitos

Durante a elicitação, você reúne uma ampla coleção de requisitos de diferentes fontes, com diferentes técnicas e em diferentes níveis de abstração e detalhe. As técnicas de elicitação que você utiliza não garantem por si só que esta coleção como um todo forme um conjunto único, consistente e acordado de requisitos que captura a essência do sistema. Tanto

durante como após a elicitação de um conjunto de requisitos para um determinado sistema, você pode descobrir que alguns dos requisitos são conflitantes: eles podem ser inconsistentes, incompatíveis, contraditórios. Pode ser que os requisitos conflitem entre si (p. ex., "todo texto deve ser preto no branco" vs "mensagens de erro devem ser vermelhas") ou que alguns stakeholders tenham uma opinião diferente sobre o mesmo requisito (p. ex., "mensagens de erro devem ser vermelhas" vs "mensagens de erro do usuário devem ser vermelhas, as demais devem ser azuis"). Como não podemos desenvolver um sistema (ou parte específica de) com base em requisitos conflitantes, os conflitos devem ser resolvidos antes que o desenvolvimento possa começar. Como Engenheiro de Requisitos, você é o responsável para garantir que todos os stakeholders cheguem a um entendimento compartilhado (ver Capítulo 2, Princípio 3) do conjunto completo de requisitos, que sejam relevantes para eles e que concordem com este conjunto.

Mas o que é um conflito? Um conflito é uma certa discordância entre as pessoas: "Uma interação entre agentes (indivíduos, grupos, organizações, etc.), onde pelo menos um agente percebe incompatibilidades entre seu pensamento/idéias/percepções e/ou sentimentos e/ou vontade e o do outro agente (ou agentes), e se sente limitado pela ação do outro" [Glas1999]. Em um conflito de requisitos, dois ou mais stakeholders têm uma opinião diferente ou mesmo contraditória em relação a um determinado requisito ou os requisitos não podem ser implementados em um determinado sistema ao mesmo tempo; ver Figura 4.9.

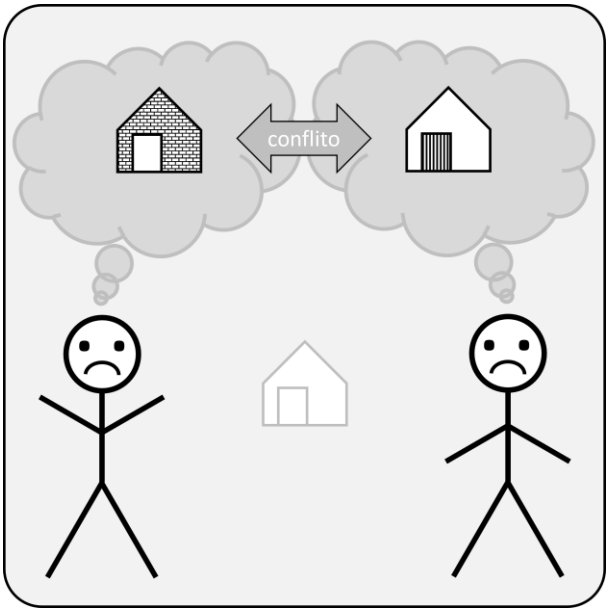


Figura 4.9 Conflito de Requisitos

Lidar com conflitos de requisitos pode ser difícil, doloroso e demorado, especialmente quando se trata de questões pessoais. Entretanto, negar ou ignorar os conflitos não é uma opção, portanto o Engenheiro de Requisitos deve procurar ativamente maneiras de resolvê-los. No final todos os stakeholders devem compreender e concordar com todos os

requisitos que são relevantes para eles. Se alguns stakeholders não concordarem, a situação deve ser reconhecida como um conflito que deve ser resolvido adequadamente.

4.3.1 Como você resolve um conflito de requisitos?

Para resolver um conflito de requisitos de forma adequada, as seguintes etapas devem ser seguidas:

- Identificação de conflitos

  Com frequência temos conflitos em nossa vida cotidiana. Eles nos geram um sentimento desagradável, então uma estratégia comum é simplesmente evitá-los, ignorá-los ou negá-los. Isso pode dificultar a identificação dos conflitos. A maioria deles tende a ser escondida e só pode ser detectada através de cuidadosa observação. Há muitos indicadores a que você pode prestar atenção, tanto na comunicação como na documentação:
  Na comunicação, você pode observar comportamentos como negação, indiferença, pedantismo, pedir continuamente por mais detalhamento, interpretações deliberadamente incorretas, ocultação ou delegação.
  Na documentação, você pode encontrar coisas como declarações contraditórias dos stakeholders, resultados conflitantes da análise de documentos ou sistemas, inconsistências em diferentes níveis de detalhes e uso inconsistente de termos.
  Se você observar tais indicadores, isto não significa necessariamente que haja um conflito de requisitos, mas você certamente deve suspeitar. Uma discussão aprofundada com os stakeholders pode então revelar um conflito oculto.

- Análise de conflitos

  Uma vez identificado um conflito, o Engenheiro de Requisitos tem que primeiro esclarecer se este conflito é ou não um conflito de requisitos. Afinal, conflitos de requisitos são a principal responsabilidade do Engenheiro de Requisitos; outros conflitos podem ser resolvidos por outros participantes, tais como um gerente de departamento ou um líder de equipe. O Engenheiro de Requisitos deve compreender plenamente a natureza do conflito de requisitos antes de tentar resolvê-lo. Isto significa que você terá que coletar mais informações sobre o conflito em si e sobre os stakeholders envolvidos.
  Muitos aspectos merecem atenção:

  - *Assunto*: o escopo, o problema ou a verdadeira questão por trás do conflito.
  - *Requisitos afetados*: quais requisitos específicos são afetados?
  - *stakeholders envolvidos*: quem discorda de quem sobre o quê?
  - *Opiniões* dos stakeholders: deixe-os expressar suas opiniões da forma mais clara possível para que todas as partes em conflito entendam a questão subjacente.
  - A *causa* do conflito: qual é a razão por trás da diferença de opiniões?
  - A *história* do conflito: o que aconteceu antes que influencia essas opiniões agora?
  - *Consequências*: os custos e riscos estimados associados tanto à resolução do conflito quanto à não resolução do mesmo.

- *Restrições do projeto*: restrições pessoais, organizacionais, de conteúdo ou de domínio podem determinar o espaço de solução.
- A análise destas informações o ajudará a reconhecer o tipo de conflito (para mais informações, consulte a Seção 4.3.2) e indicará maneiras de resolvê-lo.

- Resolução de conflitos

  Uma vez que um entendimento profundo da natureza do conflito de requisitos, da atitude dos stakeholders envolvidos e das restrições do projeto tenha sido alcançado, o Engenheiro de Requisitos selecionará uma técnica de resolução adequada. Muitas técnicas podem ser usadas, como explicado na Seção 4.3.3. O primeiro passo deve ser sempre conseguir que a técnica escolhida seja aceita pelos stakeholders envolvidos antes de aplicá-la. Se alguns stakeholders não concordarem a priori com a aplicação de uma determinada técnica, certamente não aceitarão o resultado da mesma, portanto, no final, o conflito não será resolvido. Em princípio, o Engenheiro de Requisitos não é um stakeholder envolvido no conflito, portanto você pode e deve aplicar as técnicas de resolução selecionadas de uma maneira objetiva, estritamente neutra, e acolher qualquer resultado que resulte da aplicação da técnica.

- Documentação da Resolução de Conflitos. A resolução de conflitos pode influenciar os requisitos de uma forma que não é óbvia para alguém que não esteve envolvido no conflito. O conjunto de requisitos resultante pode parecer ilógico ou ineficiente. Portanto, a resolução do conflito deve ser devidamente documentado e comunicado com relação a aspectos como os seguintes:
  - As suposições sobre o conflito e sua resolução
  - As possíveis alternativas consideradas
  - Restrições que influenciaram a escolha da técnica escolhida e/ou da resolução
  - A forma como o conflito foi resolvido, incluindo as razões para a resolução escolhida
  - Quais foram os tomadores de decisão e outros colaboradores relevantes
  - Se você não documentar a resolução, após algum tempo, os stakeholders podem simplesmente esquecer ou ignorar as decisões que foram tomadas. E mais tarde no projeto, os desenvolvedores podem não entender a lógica por trás de um determinado design de sistema e podem implementá-lo de uma maneira diferente.

Você não precisa ter medo de conflitos de requisitos, pois eles sempre ocorrerão. Isto não deve ser uma surpresa para você; na verdade, você deve se preocupar se não detectar nenhum conflito. Eles são bastante comuns, portanto, se você não os encontrar, provavelmente deixou de identificar alguns. Mas nunca os ignore. Se você não resolver imediatamente todos os conflitos de requisitos que perceber, eles aparecerão mais tarde no processo de desenvolvimento. E como Barry Boehm [Boeh1981] já descobriu há muito tempo, quanto mais tarde você descobrir um problema, mais caro será para resolvê-lo.

4.3.2 Tipos de Conflito

Para se obter uma melhor compreensão da natureza de um conflito, é útil distinguir entre diferentes tipos de conflito. Isto ajuda na seleção de técnicas de resolução adequadas.

Discernimos seis tipos de conflito:

- Conflito de temas

  Um *conflito de conteúdo* ocorre quando as partes em conflito realmente têm diferentes necessidades factuais, causadas principalmente pelo uso pretendido do sistema em ambientes diferentes. Um bom exemplo é um sistema que deve ser utilizado em diferentes países, cada um com sua própria legislação. Pode ser difícil resolver tal conflito porque os fatos subjacentes não podem ser alterados. A primeira coisa a fazer então é analisar e documentar estes fatos em detalhe e fazer com que as partes em conflito concordem sobre a natureza exata do conflito.

- Conflito de dados

  Um *conflito de dados* está presente quando algumas partes se referem a dados inconsistentes de fontes diferentes ou interpretam os mesmos dados de uma maneira diferente. Isto pode ser devido à má comunicação, falta de dados complementares, diferenças culturais, preconceitos existentes etc. As estimativas em particular, como as vendas futuras, podem facilmente gerar um conflito de dados, já que muitas vezes são baseadas em suposições. Detectar um conflito de dados não é fácil, porque, como Engenheiro de Requisitos, você pode pensar que suas próprias fontes estão certas e sua própria interpretação é auto evidente. Devido a este viés, você frequentemente suspeita de outro tipo de conflito no início. Entender como as pessoas podem chegar a uma interpretação diferente requer muita empatia. A comunicação – repetidamente – é a chave tanto para detectar quanto para resolver este tipo de conflito.

- Conflito de interesses

  Um *conflito de interesses* é baseado em diferentes posições das partes em conflito, formado por objetivos pessoais, objetivos relacionados a um grupo, ou objetivos relacionados a um papel. Você deve compreender as preocupações e necessidades dos stakeholders envolvidos antes de poder resolver este tipo de conflito. Entretanto, tenha em mente que, no caso de interesses pessoais, os stakeholders frequentemente não revelam seus verdadeiros motivos e apresentam argumentos aparentemente factuais, mas essencialmente artificiais. Se uma discussão é sobre um conflito de interesses, você pode observar as partes em conflito tentando convencer umas às outras a seguir seus argumentos e aceitar as necessidades do papel ou do grupo que representam. A resolução pode se beneficiar da identificação e do fortalecimento dos interesses comuns. Trabalhar em um entendimento comum sobre os ganhos e as dores de ambas as partes pode ser um ponto de partida para encontrar uma solução.

- Conflito de valor

  Um *conflito de valores* é baseado em diferenças de valores e princípios dos stakeholders envolvidos. Comparado a um conflito de interesses, um conflito de valores é mais individual e relacionado a perspectivas de valores globais e de longo prazo. Os valores são mais estáveis do que os interesses e raramente mudam no curto prazo. Se um conflito de valores for o motivo de uma discussão, as partes em conflito enfatizarão porque seus argumentos são importantes do ponto de vista delas, revelando seus valores e princípios interiores. Eles tendem a insistir em seus argumentos e relutam em desistir. Para resolver tais conflitos, procure valores mais elevados que unam as partes. Conflitos de valores são notoriamente difíceis de resolver, de alcançar uma compreensão mútua e o melhor que você pode conseguir é o reconhecimento dos princípios de cada um.

- Conflito de relacionamento

  Um *conflito de relacionamento* é geralmente baseado em experiências negativas com outra parte no passado, ou em situações comparáveis com pessoas semelhantes. Muitas vezes, emoções e mal-entendidos estão envolvidos, o que torna o conflito muito mais difícil de resolver. As partes em conflito usam indevidamente as discussões sobre requisitos para expressar sua raiva com o comportamento um do outro, esquecendo-se de fatos, números e da equidade. Trazer a discussão de volta aos requisitos raramente ajudará; às vezes, unir as partes em torno de um valor mais alto é bem-sucedido. Na maioria dos casos, você terá que escalar a questão para outros stakeholders ou para um nível superior de autoridade; substituir pessoas é uma resolução em potencial. Esteja ciente de que um conflito de relacionamento muitas vezes ocorre com outros tipos de conflito – por exemplo, um conflito de interesses. Analisar a causa raiz e resolver o outro tipo de conflito pode então ser a melhor maneira de melhorar o relacionamento.

- Conflito estrutural

  Um *conflito estrutural* envolve desigualdade de poder, competição sobre recursos limitados, ou dependências entre as partes. O desequilíbrio resultante (muitas vezes percebido por apenas uma das partes) causa problemas na comunicação e na tomada de decisão. Outra razão para tais conflitos podem ser restrições de recursos ou dependências de produtos de trabalho a serem entregues por outra parte. As partes podem usar a discussão sobre os requisitos para mudar ou preservar o status quo. O poder hierárquico pode ser utilizado para forçar decisões. Também para conflitos de alçada, muitas vezes é necessário escalar a questão para outros stakeholders ou para níveis mais altos de autoridade.

A maioria dos conflitos de requisitos pode ser categorizada como um conflito de conteúdo, dados, interesses ou valores. Conflitos de relacionamento e de alçada muitas vezes não estão diretamente relacionados aos requisitos e, portanto, o Engenheiro de Requisitos pode não ser a parte apropriada para resolvê-los. Entretanto, na realidade, a maioria dos conflitos se enquadra em mais de uma categoria pois as diferentes causas interagem. Portanto, é

aconselhável prestar atenção a todos os tipos de conflitos, mesmo que a solução não esteja dentro de sua responsabilidade. Se outra pessoa deve resolver o conflito, certifique-se de que isso aconteça; enquanto um conflito não for resolvido, ele continuará a ter um impacto negativo em seu trabalho como Engenheiro de Requisitos.

4.3.3 Técnicas de Resolução de Conflitos

Dependendo do tipo e do contexto (stakeholders, restrições etc.) de um conflito, é selecionada uma técnica de resolução adequada. As técnicas comumente utilizadas incluem [PoRu2015]:

Acordo (Agreement)

Um *acordo* resulta de uma discussão entre os stakeholders envolvidos, a ser continuada até que eles compreendam completamente as posições de cada um e concordem com uma certa opção de comum acordo. Isto pode consumir muito tempo, especialmente quando várias partes estão envolvidas. Se for bem-sucedido, ele proporcionará motivação adicional aos stakeholders, de modo que o resultado tem uma boa chance de ser duradouro. Esforçar-se para chegar a um acordo é comum em conflitos de dados. Se esta técnica não for bem-sucedida dentro de um prazo aceitável, outras técnicas podem ser utilizadas posteriormente.

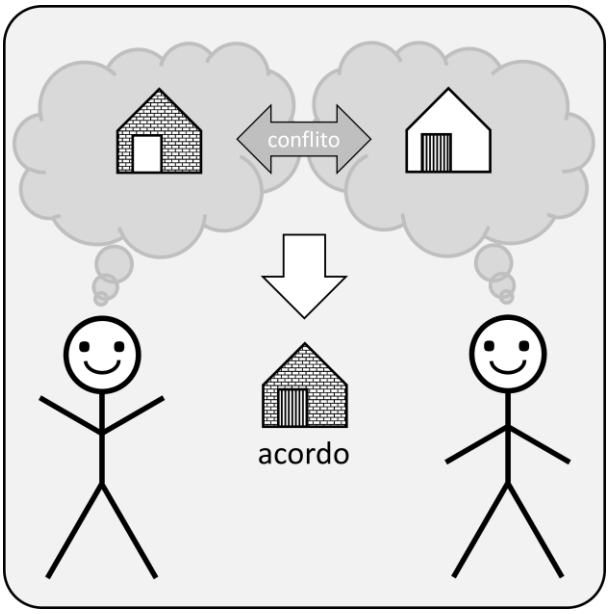


Figura 4.10 Acordo (Agreement)

Compromisso (Compromise)

Um *compromisso* é bastante semelhante a um acordo. Aqui, porém, os stakeholders concordam com uma opção que não é a preferência deles, mas com a qual podem conviver, pois aceitar o compromisso é considerado melhor do que continuar o conflito. Portanto, um compromisso também pode ser duradouro. O compromisso pode conter novos elementos inexistentes nas preferências originais dos stakeholders e que foram sugeridas pelo Engenheiro de Requisitos. Um bom compromisso é uma alternativa em que todas as partes se sentem confortáveis com o equilíbrio entre desistir de coisas e receber algo mais em troca. Quando não conseguir um acordo, tente obter um compromisso. É adequado para conflitos de conteúdo e também pode funcionar para conflitos de interesse e conflitos de alçada.

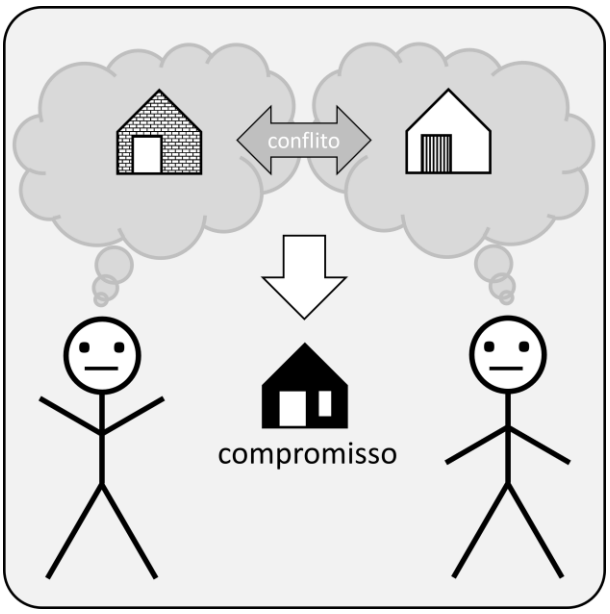


Figura 4.11 Compromisso (Compromise)

Votação (Voting)

A *votação* funciona melhor quando uma escolha relativamente simples deve ser feita entre um conjunto claro de requisitos conflitantes. Os stakeholders que participam da votação (geralmente não apenas as partes em conflito, mas todos os stakeholders envolvidos) devem compreender plenamente as alternativas e as consequências de seu voto. Para evitar influências de dependências ou um desequilíbrio de poder, a votação é melhor feita anonimamente e com um moderador neutro. O próprio procedimento de votação deve ser acordado entre os stakeholders antes da votação propriamente dita. A votação é um meio rápido e fácil de resolução de conflitos, mas a parte que perder o voto ficará desapontada e poderá precisar de atenção. A votação pode funcionar para a maioria dos tipos de conflito e pode ser uma boa maneira de resolver conflitos de conteúdos e interesses.

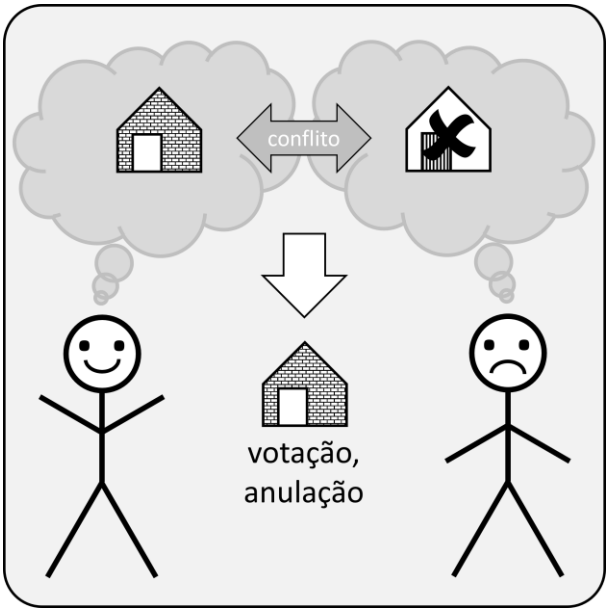


Figura 4.12 Votação (Voting)

Imposição (Overruling)

Se um acordo ou um compromisso não puder ser alcançado e pelo menos uma das partes conflitantes se recusar a participar da votação, a *imposição* pode ser uma opção. É frequentemente aplicada sob pressão, quando não há tempo suficiente para utilizar técnicas mais convenientes. Normalmente, a imposição é feita transferindo a escolha entre requisitos conflitantes para um tomador de decisão que é superior em autoridade ou hierarquia do que todas as partes conflitantes e tem poder suficiente para que a decisão seja implementada. Portanto, é uma boa maneira de resolver conflitos de interesses e de alçada. Nesta situação, é particularmente importante que o tomador de decisão compreenda plenamente as alternativas, a posição das partes em conflito e as consequências da sua decisão. Uma variante da imposição é terceirizar a decisão para, por exemplo, um especialista externo. Nesse caso, é importante primeiro obter um acordo entre os stakeholders quanto ao tomador de decisão. Assim como na votação, o *perdedor* pode precisar de atenção.

Análise de alternativas

A *definição de variantes* é frequentemente considerada para conflitos de conteúdo, de interesses e de valores. Vimos que não podemos implementar requisitos conflitantes em um mesmo sistema. A definição de variantes significa que construímos soluções separadas para todos os requisitos conflitantes. Isto geralmente é implementado através do desenvolvimento de um sistema que pode ser configurado através de parâmetros para exibir as features desejadas. Isto pode parecer uma solução perfeita, mas tem um preço: leva mais tempo para definir a solução e uma complexidade crescente (assim como custos adicionais) é introduzida no sistema, tanto para desenvolvimento como durante a operação e manutenção. Esta técnica só é viável, portanto, se houver tempo e orçamento disponíveis.

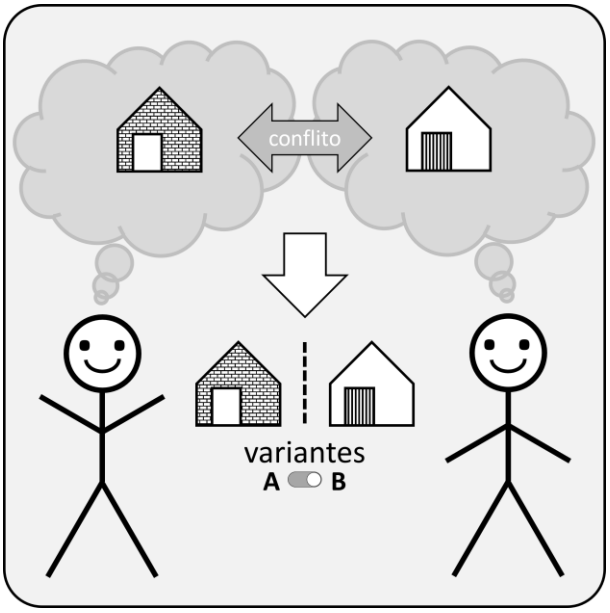


Figura 4.13 Análise de alternativas

Técnicas Auxiliares

Além disso, existem várias *técnicas auxiliares* de resolução de conflitos que normalmente não são utilizadas por conta própria, mas sim para auxiliar as técnicas acima mencionadas.

Na técnica *Considerar –All–Facts* (CAF), você considera soluções alternativas por um número de critérios pré–definidos – por exemplo, custo, tempo, risco, recursos disponíveis. Pesar estes critérios pode proporcionar mais clareza sobre os prós e contras das alternativas e ajudar a identificar a *melhor* alternativa.

A técnica Plus–Minus–Interesting (PMI, ver [DeBo2005]) é uma ferramenta de brainstorming e tomada de decisão. Ela encoraja o exame de ideias e conceitos a partir de mais de uma perspectiva e, portanto, é valiosa para a resolução de conflitos. No PMI, os participantes (geralmente todos os stakeholders envolvidos) primeiro identificam todos os aspectos positivos (+) das alternativas, depois os negativos (–), e finalmente os pontos interessantes, coisas que precisam de mais investigação. A alternativa com o maior número de vantagens e o menor número de desvantagens é a alternativa preferida.

As técnicas CAF e PMI são variantes da *matriz de decisão*, uma abordagem *metódica* para a resolução de conflitos. Os requisitos conflitantes são avaliados com base em um número (maior) de critérios, que são pontuados e utilizadas para calcular uma pontuação final (ponderada) para as alternativas. A pontuação *mais alta* vence, como a *Alternativa 1* no exemplo de Tabela 4.1 abaixo. Na verdade, a priorização (ver Seção 6.8) é aqui utilizada como uma técnica de resolução. Como dito anteriormente, estas técnicas são geralmente vistas como auxiliares: elas geram mais informações sobre alternativas e assim ajudam no uso da técnica de resolução escolhida. Eles podem até mesmo ser usados como uma única técnica se todos os stakeholders envolvidos concordarem em aceitar o resultado.

Tabela 4.1: Exemplo de uma Matriz de Decisão

		Alternativa 1: somente iOS		Alternativa 2: Android e iOS	
Critério	Peso	Pontos	Ponderado	Pontos	Ponderado
Base de clientes	2	3	6	4	8
Custo de desenvolvimento	1	3	3	2	2
Prazo de Entrega	3	4	12	2	6
Reputação	2	2	4	4	8
UX	1	5	5	3	3
Total			30		27

4.4 Validação de Requisitos

No Capítulo 2, Princípio 6, enfatizamos a importância de validar os requisitos para evitar stakeholders insatisfeitos. Porque os requisitos formam o ponto de partida para o desenvolvimento do sistema, devemos assegurar sua qualidade a montante para reduzir o desperdício de esforços a jusante, tanto no nível dos requisitos individuais quanto dos Produtos de Trabalhos que os contem (Figura 4.14).

Na nossa documentação devemos validar a cobertura das necessidades dos stakeholders, o grau de concordância entre todos os stakeholders e a verossimilhança de nossas suposições sobre o contexto do sistema antes de entregarmos os requisitos aos desenvolvedores ou fornecedores. Embora o nível de detalhes possa variar, isto se aplica tão bem para abordagens iterativas quanto para abordagens de desenvolvimento sequencial.

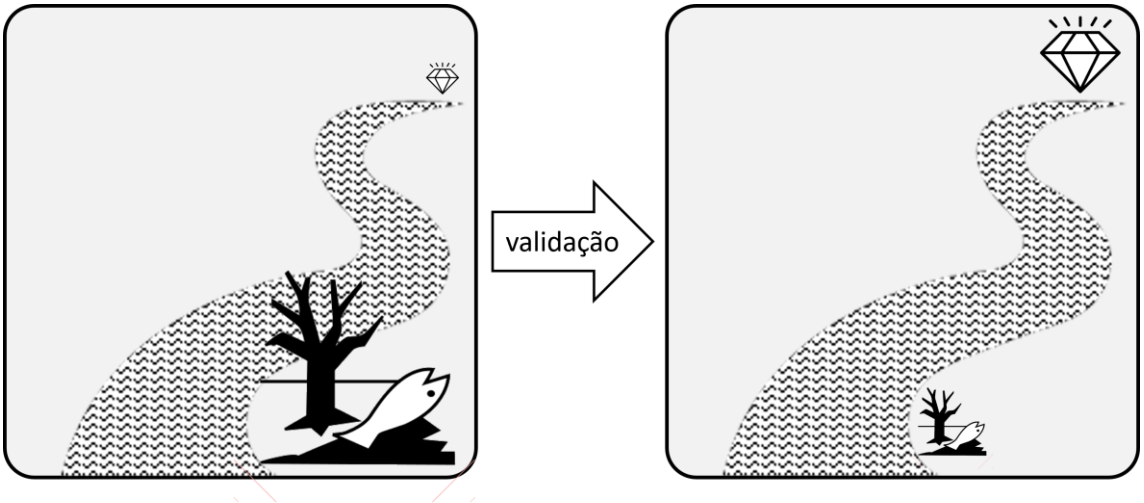


Figura 4.14 A Qualidade a Montante Reduz o Desperdício a Jusante

A validação adiciona tempo e custo ao projeto, portanto sua eficiência e eficácia devem ser uma preocupação do Engenheiro de Requisitos. Portanto, é importante monitorar e analisar continuamente os defeitos que ocorrem durante o desenvolvimento e a operação. Se a causa raiz de tais defeitos parece estar nos requisitos, o processo de validação dos requisitos falhou de alguma forma. Portanto, como Engenheiro de Requisitos, você deve procurar contínua e ativamente oportunidades para melhorá-lo.

4.4.1 Aspectos importantes para a validação

Em relação ao conceito de validação, certos aspectos são importantes para dele obter o valor máximo (ver também [PoRu2015]):

- Envolver os stakeholders corretos

  Como Engenheiro de Requisitos, você precisa decidir quem você quer convidar para participar da validação. A este respeito, um aspecto importante a ser considerado é o grau de independência entre as pessoas envolvidas na elicitação dos requisitos e daqueles que os validam. Um baixo nível de independência (convidando os stakeholders que já participaram da elicitação) é barato e fácil de organizar, mas pode ignorar certos defeitos por causa de algum foco, falta de percepção, conflitos de interesse, ou suposições errôneas dessas pessoas. Um maior grau de independência (por exemplo, convidando revisores ou auditores externos) leva mais tempo e esforço para organizar e executar e traz maiores custos (no início), mas pode, a longo prazo, ser mais eficaz para encontrar defeitos cada vez mais graves. Consequentemente, riscos maiores no escopo do projeto e/ou no contexto do sistema pedem um grau maior de independência.

- Separar a identificação e a correção de defeitos

  Pode ser tentador consertar cada defeito assim que detectado. No entanto, isso geralmente não é uma maneira eficiente nem eficaz de trabalhar, pois os defeitos

podem influenciar uns aos outros. Um defeito encontrado mais tarde durante a validação pode invalidar a correção de um defeito anterior. Um requisito inicialmente marcado como defeituoso pode provar ser correto quando todos os requisitos tiverem sido validados. Caso o esforço associado à correção do conjunto total de defeitos for muito grande você pode decidir não corrigir alguns destes defeitos (menores). E, afinal, as pessoas envolvidas na validação dos requisitos devem se concentrar em encontrar defeitos e não em sugerir ideias sobre como corrigi-los. Portanto, a recomendação é selecionar (um conjunto coerente de) requisitos para validar e também decidir quais corrigir, ou não, ao final da validação.

- Validar a partir de diferentes perspectivas

  Uma validação adequada é sempre um esforço de grupo, não uma atividade realizada pelos Engenheiros de Requisitos por conta própria. Os melhores resultados são alcançados quando a validação é realizada por uma equipe interdisciplinar na qual os participantes selecionados contribuem com sua própria experiência. Em geral, podemos dizer que a entrada, a saída e os pares devem estar representados. Em projetos iterativos, a equipe ágil é uma escolha razoável, mas o grau de independência pode ser baixo e validadores adicionais devem ser convidados; em projetos sequenciais e orientados por planejamento, uma equipe específica pode ser composta para cada esforço de validação separado. Dependendo da fase do projeto, a contribuição da área de negócio, usuários, desenvolvedores, testadores, operadores e gerentes de aplicações é útil; às vezes, podemos convocar especialistas no assunto ou especialistas em tópicos como desempenho, segurança e usabilidade.

- Validações Adicionais

  Em projetos sequenciais, a maioria dos requisitos são elicitados e documentados na fase inicial e validados minuciosamente no final dessa fase. Entretanto, este não deve ser o único momento de validação. Durante o restante do projeto, novos insights podem levar à atualização do conjunto original de requisitos que podem ser mais detalhados e expandidos. Isto pode ameaçar a qualidade, coerência e consistência dos requisitos e, portanto, validações adicionais podem ser necessárias. Estes são frequentemente planejados nos marcos de projetos.
  Em projetos iterativos, muitos dos rituais ágeis incluem esforços de validação. O planejamento de sprint, refinamento do backlog, revisões de sprint e até mesmo os stand-ups diários oferecem oportunidades para validar e melhorar os requisitos. Entretanto, esses esforços muitas vezes se concentram em requisitos individuais, detalhados e a visão do todo pode ser prejudicada. Uma validação inicial do backlog completo do produto no início de um projeto ou incremento é um bom começo. Outras iniciativas úteis são a repetição de sprints de reforço e a validação global adicional no momento do lançamento.

4.4.2 Técnicas de Validação

Quanto a outras técnicas, o Engenheiro de Requisitos pode escolher entre uma grande caixa de ferramentas de técnicas de validação que diferem em formalidade e esforço. Muitos fatores influenciam a seleção dessas técnicas – por exemplo, o ciclo de vida do desenvolvimento de software, a maturidade do processo de desenvolvimento, a complexidade e o nível de risco do sistema, requisitos legais ou regulatórios e a necessidade de uma trilha de auditoria.

Muitas vezes, no decorrer de um projeto, o grau de esforço e formalidade aumenta quanto mais perto da entrega estivermos, uma vez que decisões finais sobre o sistema e sua implementação deverão ser tomadas. Além disso, você verá que a quantidade, o valor e o nível de detalhes do feedback dos stakeholders aumentam à medida que os produtos de trabalho a serem validados se tornam mais concretos e detalhados. Isto implica na aplicação de diferentes técnicas de validação em diferentes etapas do projeto. No início de um projeto, ciclos de validação e feedback frequentes, curtos e simples, são preferíveis, como é usual em abordagens ágeis. Isto garante qualidade desde o início. Mais tarde no projeto, prevalecerão técnicas mais formais e demoradas.

Além disso, você também pode observar uma mudança no foco das atividades de validação. Nas fases iniciais de um projeto, as técnicas são usadas principalmente para validar a *especificação* dos requisitos. Em fases posteriores, o foco das mesmas técnicas pode mudar para a validação de sua *implementação*.

Em geral, podemos discernir três categorias de técnicas de validação (ver Figura 4.15):

- Técnicas de Revisão
- Técnicas Exploratórias
- Desenvolvimento de Amostras

As técnicas de revisão e de amostragem são chamadas de estáticas, pois se concentram na análise das especificações de um sistema sem executá-lo. Em técnicas exploratórias, a validação se concentra no comportamento real (ou simulado) do sistema em operação; estas técnicas são chamadas dinâmicas.

Figura 4.15 Categorias de Técnicas de Validação

A característica comum das *técnicas de revisão* é que elas consistem no estudo visual de produtos de trabalho iniciais e intermediários. Elas variam de informais a muito formais e podem ser aplicadas desde o início de um projeto até a implantação do sistema. Na maioria dos casos, a revisão dos requisitos é limitada às fases iniciais (a montante) de um projeto. Normalmente, em uma revisão, verificamos produtos de trabalho *estáticos* que definem ou descrevem como o sistema deve funcionar. Para mais informações sobre revisão, veja [OleA2018].

As *revisões informais* geralmente seguem o ciclo autor-revisor. Um autor envia um produto de trabalho a um grupo de pessoas com o pedido de validá-lo. Normalmente, este é um pequeno grupo de membros da equipe, colegas e/ou usuários envolvidos no projeto. Os autores podem selecionar o grupo por si ou sua composição pode ser prescrita pelos regulamentos da empresa. Após um curto (mas frequentemente não pré-definido) período, o autor coleta todos os comentários de revisão e os utiliza para atualizar o produto de trabalho em questão. É uma boa prática documentar os comentários em um registro de revisão e acompanhar como são tratados. Entretanto, devido à natureza informal deste tipo de revisão, os autores são livres para decidir se e como usar os comentários. Frequentemente, a revisão é repetida em várias versões intermediárias até que o autor esteja satisfeito com a sua qualidade.

Como são informais, você pode esperar poucos benefícios desses tipos de revisões para validar e melhorar a qualidade dos requisitos. Entretanto, se todos os participantes estão comprometidos com a qualidade e são capazes e dispostos a gastar tempo suficiente no processo de revisão, as revisões informais são um meio de validação fácil, barato e acessível. Na verdade, esta abordagem é comum para os rascunhos iniciais. Para a versão final de um produto de trabalho, uma técnica mais formal pode ser uma escolha melhor.

As revisões formais seguem uma forma prescrita de trabalho. Elas são frequentemente utilizadas para produtos de trabalho importantes ou em marcos, para versões finais e em

situações em que altos riscos estão em jogo. Embora existam muitos tipos de revisões formais, elas podem ser divididas em dois grupos principais:

- Walkthrough

  A essência de um *walktrough* é que o autor de um produto de trabalho o explica passo a passo a um público em uma sessão interativa. Na prática, os walkthroughs vêm em duas variantes, onde (1) os revisores participam da reunião sem qualquer preparação e ouvem o autor, fazendo perguntas ad hoc; ou (2) eles obtêm o produto do trabalho antes da reunião e preparam perguntas para o autor. Os participantes da audiência podem fazer comentários, identificar falhas e sugerir alternativas. às ideias originais. O autor dá mais explicações se necessário e pode discutir soluções para os pontos fracos identificados e avaliar as alternativas em relação às ideias originais. Há duas ocasiões em que os walkthroughs são melhor aplicados: (a) em uma fase inicial do projeto para discutir a viabilidade de um determinado conceito de sistema ou esboço de solução; e (b) na transferência de um produto de trabalho intermediário para outra parte que o utilizará como insumo para o desenvolvimento subsequente. Em projetos iterativos, os walktroughs estão principalmente presentes na forma de sessões regulares de refinamento antes de uma iteração e revisões de sprint no final da mesma.

- Inspeção

  As inspeções estão entre as técnicas de revisão mais formais. Aqui, a responsabilidade pela revisão não é do autor, mas de um líder de revisão independente, muitas vezes chamado de *moderador*. Uma inspeção é normalmente realizada na forma de uma reunião com o moderador, o autor, e um grupo de *inspetores*.
  Os inspetores são selecionados entre pares, o negócio, usuários e/ou especialistas. Eles são solicitados a verificar o produto de trabalho com base em sus expertise específica, verificar sua aderência a padrões, normas e regulamentos aplicáveis e para avaliá-lo em relação aos objetivos acordados. Muitas vezes, esta verificação pelos inspetores é realizada durante sua preparação individual antes da reunião propriamente dita, guiada por checklists detalhados. Na reunião de revisão, o autor participa como ouvinte, explicando coisas que não estão claras e tentando entender os comentários dos inspetores e as consequências para o produto do trabalho. Tipicamente, uma inspeção segue um processo rigoroso e documentado que é gerenciado pelo moderador e se concentra em encontrar defeitos, medir aspectos de qualidade definidos e fornecer uma trilha de auditoria detalhada. Nesta forma, as inspeções são frequentemente usadas para decidir sobre a liberação de um produto de trabalho para uma próxima etapa do processo de desenvolvimento, ou mesmo para a implementação final. As inspeções são aplicadas principalmente em sistemas críticos (segurança) e processos de negócios. Em abordagens ágeis, esta forma

formal de revisão é incorporada na própria metodologia – por exemplo, com as cerimônias Scrum (refinamento, planejamento, revisão da sprint).

As *técnicas exploratórias* oferecem a um grupo de stakeholders e potenciais usuários a oportunidade de ganhar experiência prática com uma versão intermediária do (ou parte do) sistema em desenvolvimento. Em contraste com as revisões, as técnicas exploratórias são *dinâmicas*: elas observam o comportamento (real ou simulado) do sistema em funcionamento como experimentado pelos usuários através das interfaces de usuário. Os participantes são convidados a utilizar o sistema de uma forma semelhante ao uso pretendido em produção. Eles são relativamente livres para fazê-lo, às vezes certas orientações são fornecidas. Após um período de uso, os participantes relatam suas experiências e seu feedback sobre o comportamento atual do sistema ao Engenheiro de Requisitos. Isto pode incluir defeitos encontrados e sugestões para melhorias.

As técnicas exploratórias são comuns nas abordagens iterativas e de Design Thinking. De fato, o desenvolvimento incremental usual, começando com o lançamento de um *produto mínimo viável (MVP),* seguido pela adição paulatina e gradativa de mais funcionalidades, medindo cuidadosamente as reações do mercado e ajustando o sistema consequentemente, pode ser visto como uma validação exploratória dos requisitos em produção.

As técnicas exploratórias comuns incluem:

- Protótipos

  Na validação via *protótipos*, uma versão inicial específica do sistema é dada a um grupo de stakeholders para avaliação. Esta versão pode ser explicitamente construída para fins de validação, após a qual é descartada; chamamos isto de um protótipo exploratório ou descartável. É claro que os protótipos evolutivos, que são continuamente atualizados e ampliados até chegarem ao produto final, também podem ser usados para validação durante seu desenvolvimento. A essência de qualquer protótipo visto de fora, se pareça com o sistema pretendido, permitindo aos stakeholders ganhar experiência prática enquanto a estrutura interna ainda pode estar inacabada, inoperante, ou mesmo completamente ausente. Ao utilizar um protótipo para validação, você pode tê-lo construído para verificar uma característica específica, tal como interface de usuário, segurança ou desempenho.

- Elicitação e Validação se Complementam

  Como vimos na Seção 4.2.3, a prototipação e o storyboarding também podem ser usados como técnicas de elicitação. De fato, estas técnicas suportam tanto a elicitação quanto a validação, andando de mãos dadas: ao validar os requisitos elicitados em um momento anterior, é quase certo que você detectará novos requisitos no feedback dos participantes. Ambos os aspectos da prototipação são muito proeminentes nas abordagens de Design Thinking (ver [LiOg2011]).

- Testes alfa e beta

  Nos testes alfa e nos testes beta, uma versão de pré-produção totalmente funcional do sistema é fornecida aos usuários finais para operação com os processos comerciais pretendidos em um ambiente realista.
  Os *testes Alfa* são feitos no site do desenvolvedor em um ambiente simulado. O grupo de participantes é relativamente pequeno, algumas orientações podem ser dadas, e é possível observar a interação dos usuários com o sistema – por exemplo, em um laboratório de usabilidade.
  Os *testes Beta* são realizados nas instalações do usuário final em produção real (ou em qualquer ambiente que o usuário final prefira). O sistema é oferecido (em sua maioria gratuitamente) a um grupo de usuários (às vezes selecionados, mas geralmente desconhecidos), com a solicitação implícita para validar sua aparência e comportamento. Nos testes beta, é importante estimular todos os participantes a dar seu feedback e fornecer uma maneira fácil de fazê-lo. A análise deste feedback após um período prolongado de uso pode dar pistas valiosas sobre a qualidade dos requisitos. É particularmente útil para verificar certas suposições feitas durante a elicitação e o desenvolvimento.

- Testes A/B

  Os *testes A/B* são frequentemente realizados com uma versão liberada do sistema em um ambiente operacional, mas também podem ser disponibilizados na forma de versões de pré-lançamento em um ambiente de teste protegido. A essência dos testes A/B é que o sistema é oferecido a diferentes grupos de usuários (usualmente selecionados aleatoriamente) em duas variantes que diferem em design e/ou funcionalidade e realizam os objetivos do usuário de maneira distinta. A reação dos dois grupos é medida e comparada; isto funciona melhor quando os grupos são suficientemente grandes para permitir análises estatísticas. A análise dará então informações sobre a qualidade dos requisitos subjacentes e sobre a exatidão das suposições anteriores. Os testes A/B têm um papel proeminente em *The Lean Startup*, uma das abordagens do Design Thinking (ver [Ries2011]).

No *desenvolvimento de amostras*, você fornece um conjunto de requisitos como entrada para os desenvolvedores; eles tentam produzir alguns produtos de trabalho intermediários usuais (por exemplo, projetos, códigos, casos de teste, manuais) com base nesta entrada. O sistema em si não é operacional (ainda), portanto, este tipo de validação é *estática*, assim como na revisão. Durante este esforço, os desenvolvedores podem detectar falhas, como falta de clareza, omissões e inconsistências que os impedem de produzir os resultados pretendidos. Naturalmente, estas falhas serão corrigidas. No entanto, a quantidade e a gravidade das falhas detectadas é uma indicação da qualidade dos requisitos. Se esta qualidade não for suficiente, é necessária mais validação – por exemplo, revisões adicionais.

Uma validação semelhante pode ser realizada pelos próprios Engenheiros de Requisitos. Nesse caso, você tenta documentar um conjunto de requisitos em uma forma de representação diferente do tipo original: geralmente, convertendo uma especificação de requisitos criada em linguagem natural em um modelo relevante, ou um modelo específico

em uma descrição textual. Este exercício é especialmente útil para detectar omissões. Se você encontrar sérios problemas nesta conversão, isto indica a necessidade de uma validação adicional.

4.5 Leitura adicional

Glinz e Wieringa [GlWi2007] explicam a noção e a importância dos stakeholders. Alexander [Alex2005] discute como classificar os stakeholders. Bourne [Bour2009] lida com a gestão dos stakeholders. Lim, Quercia e Finkelstein [LiQF2010] investigam o uso de redes sociais para análise dos stakeholders. Humphrey [Hump2017] discute personas de usuários.

Zowghi e Coulin [ZoCo2005] apresentam uma visão geral das técnicas de elicitação de requisitos. Gottesdiener [Gott2002] escreveu um clássico livro didático sobre workshops na ER. Carrizo, Dieste e Juristo [CaDJ2014] investigam a seleção de técnicas adequadas de elicitação.

Maalej, Nayebi, Johann e Ruhe [MNJR2016] discutem o uso de feedback explícito e implícito do usuário para elicitar requisitos. Maiden, Gitzikis e Robertson [MaGR2004] discutem como a criatividade pode fomentar a inovação na ER.

O livro de Moore [Moor2014] é um clássico sobre gestão de conflitos. Glasl [Glas1999] discute como lidar com conflitos. Grunbacher e Seyff [GrSe2005] discutem como chegar a um acordo através da negociação de requisitos ao validar requisitos ou resolver conflitos.

A validação é coberta em qualquer livro didático sobre a ER; ver [Pohl2010], por exemplo.

5 Processo e Estrutura de Trabalho

Sempre que o trabalho tem que ser feito de forma sistemática, um *processo* é necessário para dar contorno e estruturar a forma de trabalho e a criação de produtos de trabalho.

> **Definição 5.1. Processo:**
> Um conjunto de atividades inter-relacionadas realizadas em uma determinada ordem para processar informações ou materiais.

Um processo de Engenharia de Requisitos (ER) organiza como as tarefas da ER são realizadas utilizando práticas apropriadas e produzindo os produtos de trabalho necessários. Entretanto, não há nenhum processo de ER comprovado, de tamanho único (ver Seção 1.4). Consequentemente, os Engenheiros de Requisitos têm que configurar um processo de ER sob medida que se adapte à situação em questão.

O processo de ER molda o fluxo de informações e o modelo de comunicação entre os participantes envolvidos na ER (por exemplo, clientes, usuários, Engenheiros de Requisitos, desenvolvedores e testadores). Também define os produtos de trabalho da ER a serem utilizados ou produzidos. Um processo de ER adequado fornece a estrutura na qual os Engenheiros de Requisitos elicitam, documentam, validam e gerenciam os requisitos.

Neste capítulo, você aprenderá sobre os fatores que influenciam o processo de ER e como configurar um processo apropriado a partir de um conjunto de facetas de processo.

5.1 Fatores de Influência

Há uma variedade de fatores de influência a serem considerados na configuração de um processo de ER. Antes de começar com a configuração de um processo de ER, estes fatores precisam ser investigados e analisados.

Por um lado, tal análise fornece informações sobre como configurar o processo de ER. Por exemplo, quando a análise indica que os stakeholders têm apenas uma vaga ideia sobre seus requisitos, deve ser escolhido um processo de ER que apoie a exploração dos mesmos. Por outro lado, os fatores de influência também limitam o espaço de possíveis configurações de processos. Por exemplo, se os stakeholders estiverem disponíveis apenas no início de um projeto de desenvolvimento de sistema, um processo que se baseie no feedback contínuo dos stakeholders não seria adequado. Abaixo, discutimos fatores de influência importantes para o processo de ER.

Adequação geral ao processo. Ao definir ou configurar um processo de ER, é vital conhecer e entender o processo de desenvolvimento escolhido para o sistema a ser desenvolvido – definir um processo de ER que não se encaixa no processo de desenvolvimento não faz sentido. O processo de desenvolvimento pode exigir produtos de trabalho que o processo de ER deve entregar. A terminologia usada para o processo de ER deve ser alinhada à

terminologia do processo de desenvolvimento. Em particular, a terminologia para os produtos de trabalho deve ser alinhada. Isto ajuda a evitar confusões e mal-entendidos. Também facilita a introdução do processo de ER, assim como o treinamento e o coaching das pessoas que têm que trabalhar de acordo com mesmo. Por exemplo, se o sistema for desenvolvido utilizando um processo sequencial, orientado por planejamento, que conta com a existência de uma especificação abrangente dos requisitos do sistema e um glossário do sistema no final da fase de requisitos, o processo de ER escolhido deve se encaixar na fase de requisitos do processo geral e produzir os dois produtos de trabalho necessários.

Contexto de desenvolvimento. O contexto de desenvolvimento também impacta o processo de ER. Coisas a considerar incluem o relacionamento cliente-fornecedor-usuário, tipo de desenvolvimento, questões contratuais e confiança. Ao analisar o contexto de desenvolvimento, algumas perguntas precisam ser respondidas:

- Relação cliente-fornecedor-usuário: Existe um cliente designado que encomenda o sistema e paga por ele e um fornecedor que desenvolve o sistema? Clientes e fornecedores fazem parte da mesma organização ou pertencem a organizações diferentes? No primeiro caso, quais pessoas atuam no papel de cliente e quais atuam como fornecedor? Quem são os usuários do sistema? Os usuários pertencem à organização do cliente?

  Se não, eles usam o sistema como um produto ou serviço para interagir com o cliente (por exemplo, em negócios eletrônicos) ou eles compram o sistema como um produto ou serviço do cliente (por exemplo, um aplicativo móvel)?

- Tipo de desenvolvimento: Qual é a estrutura organizacional para o desenvolvimento de um sistema? Os tipos típicos incluem:
  - Um fornecedor especifica e desenvolve um sistema para um cliente específico que irá utilizar o sistema.
  - Uma organização desenvolve um sistema com a intenção de vendê-lo como um produto ou serviço para muitos clientes em um determinado segmento de mercado.
  - Um fornecedor configura um sistema para um cliente a partir de um conjunto de componentes prontos para uso.
  - Um fornecedor melhora e evolui um produto existente.
- *Contrato*: Existe um contrato ou acordo similar que define formalmente os resultados, custos, prazos, responsabilidades etc.? Os contratos podem ser contratos clássicos de preço fixo entre um cliente e um fornecedor, com funcionalidade, prazos e custo fixos, ou apenas definem uma estrutura orçamentária, enquanto a funcionalidade é definida iterativamente.
- *Confiança*: As partes envolvidas confiam umas nas outras? Se, por exemplo, o cliente e o fornecedor não confiam um no outro, os requisitos têm de ser especificados com mais detalhes do que seria necessário em uma relação baseada na confiança.

Disponibilidade e capacidade dos stakeholders. A disponibilidade dos stakeholders restringe as opções de configuração para o processo de ER. Por exemplo, um processo que requer

uma interação contínua e próxima com os stakeholders não pode ser escolhido se os principais stakeholders estiverem disponíveis apenas por um curto período de tempo no início do processo.

A capacidade dos stakeholders também influencia o processo: quanto menos stakeholders forem capazes de expressar suas necessidades claramente, e quanto menos conhecerem suas necessidades reais, mais o processo de ER deverá acomodar a exploração das necessidades.

Entendimento compartilhado. Apenas pouca Engenharia de Requisitos é necessária quando há um alto grau de entendimento compartilhado (ver Capítulo 2, Princípio 3) entre os stakeholders, Engenheiros de Requisitos, projetistas e desenvolvedores sobre o problema e os requisitos. Consequentemente, quanto melhor for o entendimento compartilhado, mais leve pode ser o processo de ER [GlFr2015].

Complexidade e criticidade. O grau de detalhe a que os requisitos precisam ser especificados depende fortemente da complexidade e da criticidade do sistema a ser desenvolvido. Quando um sistema é complexo e/ou crítico com relação à segurança ou proteção, o processo de ER escolhido deve acomodar uma especificação detalhada dos requisitos críticos, incluindo modelos formais ou semiformais e forte validação – por exemplo, verificando modelos que expressam o comportamento prescrito ou construindo protótipos.

Restrições. É claro que todos estes fatores de influência limitam o espaço de possíveis configurações de processos de ER. Quando falamos de restrições, queremos dizer aquelas restrições que são explicitamente impostas, por exemplo, pelo cliente ou por uma autoridade regulatória. Tais restrições podem implicar a criação obrigatória de certos produtos de trabalho e seguir um processo obrigatório para a produção desses produtos de trabalho. Os clientes ou uma autoridade regulatória também podem exigir um processo de ER em conformidade com alguma determinada norma.

Tempo e orçamento disponíveis. Se os cronogramas e orçamentos forem apertados, o tempo e orçamento disponíveis para a ER precisam ser usados com sabedoria, o que normalmente implica na escolha de um processo de ER leve. A escolha de um processo de ER iterativo ajuda a priorizar os requisitos e a implementar os mais importantes dentro do orçamento e do cronograma determinados.

Volatilidade dos requisitos. Se muitos requisitos forem susceptíveis de mudar, é aconselhável escolher um processo de ER iterativo e amigável à mudança.

Experiência dos Engenheiros de Requisitos. O processo de ER escolhido deve corresponder às competências e experiência dos Engenheiros de Requisitos envolvidos. Caso contrário, tempo e orçamento adicionais devem ser alocados para treinar e orientar no processo escolhido. É melhor escolher um processo bastante simples que os Engenheiros de Requisitos possam lidar adequadamente do que um processo sofisticado e complicado que os sobrecarregue.

5.2 Dimensões de Processos de Engenharia de Requisitos

Definir o processo de ER a partir do zero para cada empreendimento de ER é um desperdício de esforço. Sempre que os fatores de influência o permitam, o processo deve ser configurado a partir de elementos pré-existentes. A fim de fornecer orientações sobre a forma de configurar um processo de ER adequado, descrevemos três dimensões com dois aspectos cada, juntamente com critérios de seleção a considerar para cada aspecto [Glin2019]. Mais tarde, na Seção 5.3, utilizamos estes aspecto para configurar os processos de ER. Figura 5.1 mostra uma visão geral das dimensões e dos aspectos.

Figura 5.1 Dimensões de Processos de ER

As dimensões abrangem um espaço tridimensional de opções de configuração de processos de ER. Cada dimensão vem com critérios para selecioná-los.

A aplicabilidade destes critérios decorre da análise dos fatores de influência discutidos na Seção 5.1 acima. Note que nem todos os critérios precisam ser preenchidos para escolher um aspecto de uma dimensão.

5.2.1 Dimensão de Tempo: Sequencial vs. Iterativo

A dimensão de tempo trata da organização das atividades da ER na escala temporal do processo de desenvolvimento. Distinguimos entre processos sequenciais e iterativos.

Em um *processo de ER sequencial*, os requisitos são especificados antecipadamente em uma única fase do processo. A ideia é produzir uma especificação abrangente de requisitos

que requer pouca ou nenhuma adaptação ou poucas mudanças durante o projeto e implementação do sistema. A criação de uma especificação de requisitos abrangente exige um processo também abrangente. Assim, na maioria dos casos, os processos de ER sequenciais são processos pesados (heavyweight).

Critérios para a escolha de um processo de ER sequencial:

- O processo de desenvolvimento do sistema é orientado a planejamento antecipado e, na sua maioria, sequencial.
- Os stakeholders estão disponíveis, conhecem seus requisitos e conseguem especificá-los antecipadamente.
- Uma especificação abrangente dos requisitos é necessária como base contratual para terceirizar ou licitar o projeto e a implementação do sistema.
- Autoridades regulatórias exigem uma especificação abrangente e formalmente liberada dos requisitos em uma fase inicial do desenvolvimento.

Em um *processo de ER iterativo*, os requisitos são especificados gradualmente, começando com objetivos gerais e alguns requisitos iniciais, adicionando ou modificando os requisitos a cada iteração. A ideia é entrelaçar a especificação dos requisitos com o projeto e a implementação do sistema. Devido aos rápidos feedbacks e à capacidade de acomodar mudanças ou coisas esquecidas em iterações posteriores, os processos de ER iterativos podem ser processos leves (lightweight).

Critérios para a escolha de um processo de ER iterativo:

- O processo de desenvolvimento do sistema é iterativo e ágil.
- Muitos requisitos não são conhecidos previamente, mas vão surgir e evoluir durante o desenvolvimento do sistema.
- Os stakeholders estão disponíveis de tal forma que podem ser estabelecidos pequenos ciclos de feedback como forma de mitigar o risco de desenvolver o sistema errado.
- A duração do desenvolvimento permite mais do que apenas uma ou duas iterações.
- A capacidade de mudar facilmente os requisitos é importante.

5.2.2 Dimensões de Finalidade: Prescritivo vs. Exploratório

A dimensão de finalidade trata da finalidade e do papel dos requisitos no desenvolvimento de um sistema. Distinguimos entre processos de ER prescritivos e exploratórios.

Num *processo de ER prescritivo,* a especificação dos requisitos constitui um contrato: todos os requisitos são obrigatórios e devem ser implementados. A ideia é criar uma especificação de requisitos que possa ser implementada com nenhuma ou pouca interação adicional entre os stakeholders e os desenvolvedores.

Critérios para a escolha de um processo de ER prescritivo:

- O cliente requer um contrato fixo para o desenvolvimento do sistema, muitas vezes com funcionalidade, escopo, preço e prazo fixos.
- A funcionalidade e o escopo têm precedência sobre o custo e os prazos.

- O desenvolvimento do sistema especificado pode ser licitado ou terceirizado.

Em um *processo de ER exploratório*, apenas os objetivos são conhecidos a priori, enquanto os requisitos concretos têm que ser elicitados. A ideia é que os requisitos frequentemente não são conhecidos a priori, mas têm de ser explorados.

Critérios para a escolha de um processo de ER exploratório:

- Os stakeholders inicialmente só têm uma vaga ideia sobre seus requisitos.
- Os stakeholders estão fortemente envolvidos e fornecem um feedback contínuo.
- Prazos e custos precedem a funcionalidade e o escopo.
- O cliente está satisfeito com um contrato de pagamento por serviços com metas, recursos e o preço a ser pago por um determinado período de tempo ou número de iterações.
- Não está claro a priori quais requisitos serão realmente implementados e em que ordem serão implementados.

5.2.3 Dimensão Alvo: Específico para o Cliente vs. Orientado para o Mercado

A dimensão alvo considera o tipo de desenvolvimento: que tipo de desenvolvimento visamos com o processo de ER? Distinguimos entre processos de ER específicos do cliente e processos de ER orientados para o mercado.

Em um *processo de ER específico para o cliente*, o sistema é encomendado por um cliente e desenvolvido por um fornecedor. Observe que o fornecedor e o cliente podem fazer parte da mesma organização. A ideia é que o processo de ER reflita a relação cliente-fornecedor.

Critérios para a escolha de um processo de ER específico para o cliente:

- O sistema será utilizado principalmente pela organização que encomendou o sistema e pagou pelo seu desenvolvimento.
- Os stakeholders mais importantes estão principalmente associados à organização do cliente.
- Indivíduos podem ser identificados para os papéis de stakeholders.
- O cliente quer uma especificação de requisitos que possa servir como um contrato.

Em um *processo de ER orientado para o mercado*, o sistema é desenvolvido como um produto ou serviço para um mercado, visando segmentos específicos de usuários. A ideia é que a organização que desenvolve o sistema também conduz o processo de ER.

Critérios para a escolha de um processo de ER orientado para o mercado:

- A organização que desenvolve o sistema ou um de seus clientes pretende vender o sistema em algum segmento de mercado como um produto ou serviço.
- Os usuários potenciais não são identificáveis individualmente.
- Os Engenheiros de Requisitos têm de conceber os requisitos de modo a corresponderem às necessidades previstas dos usuários visados.

- Product owners, pessoal de marketing, designers digitais e arquitetos de sistemas são os principais stakeholders.

5.2.4 Dicas e Ressalvas

É importante observar que os critérios dados acima são heurísticas. Elas não devem ser consideradas como um conjunto de regras fixas que sempre se aplicam. Por exemplo, a terceirização do desenvolvimento do sistema é feita preferencialmente com um processo de ER prescritivo e não com um processo de ER exploratório. Isto porque o contrato entre o cliente e o fornecedor é normalmente baseado em uma especificação abrangente dos requisitos. Entretanto, também é possível negociar um contrato de terceirização com base em um processo de ER exploratório.

Pode haver pré-requisitos para escolher certos aspectos das dimensões de processo ou a escolha pode acarretar consequências que devem ser consideradas. Aqui estão alguns exemplos:

- Os processos sequenciais de ER só funcionam se houver um processo sofisticado de mudança de requisitos.
- Processos sequenciais de ER implicam longos ciclos de feedback: pode levar meses ou mesmo anos desde a escrita de um requisito até que seus efeitos sejam observados no sistema implementado. Para mitigar o risco de desenvolver o sistema errado, os requisitos devem ser validados intensivamente quando se utiliza um processo de ER sequencial.
- Num processo de ER orientado para o mercado, o feedback dos usuários é o único meio de validar se o produto irá realmente satisfazer as necessidades do segmento de usuários visados.
- Em um cenário ágil, um processo de ER iterativo e exploratório se encaixa melhor. As iterações têm uma duração fixa (normalmente 2–6 semanas). O Product Owner desempenha um papel central no processo de ER, coordenando os stakeholders, organizando os produtos de trabalho da ER e comunicando os requisitos à equipe de desenvolvimento.

As três dimensões mencionadas acima não são totalmente independentes: a escolha feita por uma dimensão pode influenciar o que pode ou deve ser escolhido em outras. Aqui estão alguns exemplos:

- Os aspectos sequencial e prescritivo são frequentemente escolhidos em conjunto, o que significa que quando os Engenheiros de Requisitos decidem sobre um processo de ER sequencial, eles normalmente decidem sobre um processo que é tanto sequencial quanto prescritivo.
- Os processos de ER exploratórios são tipicamente também processos iterativos (e vice-versa).
- Um processo de ER orientado para o mercado não combina bem com um processo sequencial e prescritivo.

5.2.5 Considerações adicionais

O grau em que um processo de ER deve ser estabelecido e seguido, assim como o volume de produtos de trabalho de requisitos a serem produzidos neste processo, depende do grau de entendimento compartilhado e também da criticidade do sistema.

Quanto melhor o entendimento compartilhado e quanto menor a criticidade, mais simples e mais leve pode ser o processo de ER.

Quando há pouco prazo e orçamento disponível para a ER, os recursos disponíveis devem ser utilizados com cuidado. A escolha de um processo iterativo e explorativo ajuda. Além disso, o processo deve se concentrar na identificação e tratamento daqueles requisitos que são críticos para o sucesso do sistema.

Finalmente, o processo de ER deve se adequar à experiência dos Engenheiros de Requisitos. Quanto menores forem suas habilidades e experiência, mais simples deve ser o processo de ER – não faz sentido definir um processo sofisticado quando as pessoas envolvidas não conseguem aplicar este processo adequadamente.

5.3 Configuração de um Processo de Engenharia de Requisitos

Em um contexto concreto de desenvolvimento do sistema, os Engenheiros de Requisitos ou a(s) pessoa(s) responsáveis pela ER têm que escolher o processo de ER a ser aplicado. Recomendamos analisar primeiro os fatores de influência (ver Seção 5.1) e depois selecionar uma combinação adequada das dimensões de processo descritas na Seção 5.2.

5.3.1 Combinações Típicas de Dimensões

Três combinações de dimensões (ou suas variantes) ocorrem frequentemente na prática [Glin2019]. A seguir, descrevemos brevemente cada um deles e os caracterizamos em termos de seu principal caso de aplicação, produtos de trabalho típicos e fluxo de informação típico. Além disso, damos um exemplo. Figura 5.2 mostra as três configurações típicas de processo no espaço das três dimensões.

Processo de ER Participativo: Iterativo, Explorativo e Específico do Cliente

Um processo de ER participativo é normalmente escolhido em ambientes ágeis quando há um cliente que encomenda um sistema e uma equipe de desenvolvimento que o projeta e implementa. O foco é explorar os requisitos em uma série de iterações em estreita colaboração entre os stakeholders do lado do cliente, os Engenheiros de Requisitos e a equipe de desenvolvimento.

Aplicação principal:	Fornecedor e cliente colaboram estreitamente; os stakeholders estão fortemente envolvidos tanto na ER como nos processos de desenvolvimento.
Produtos de trabalho típicos:	Backlog do produto com histórias de usuários e/ou descrições de tarefas, protótipos
Fluxo de informação típico:	Interação contínua entre stakeholders, Product Owners, Engenheiros de Requisitos e desenvolvedores

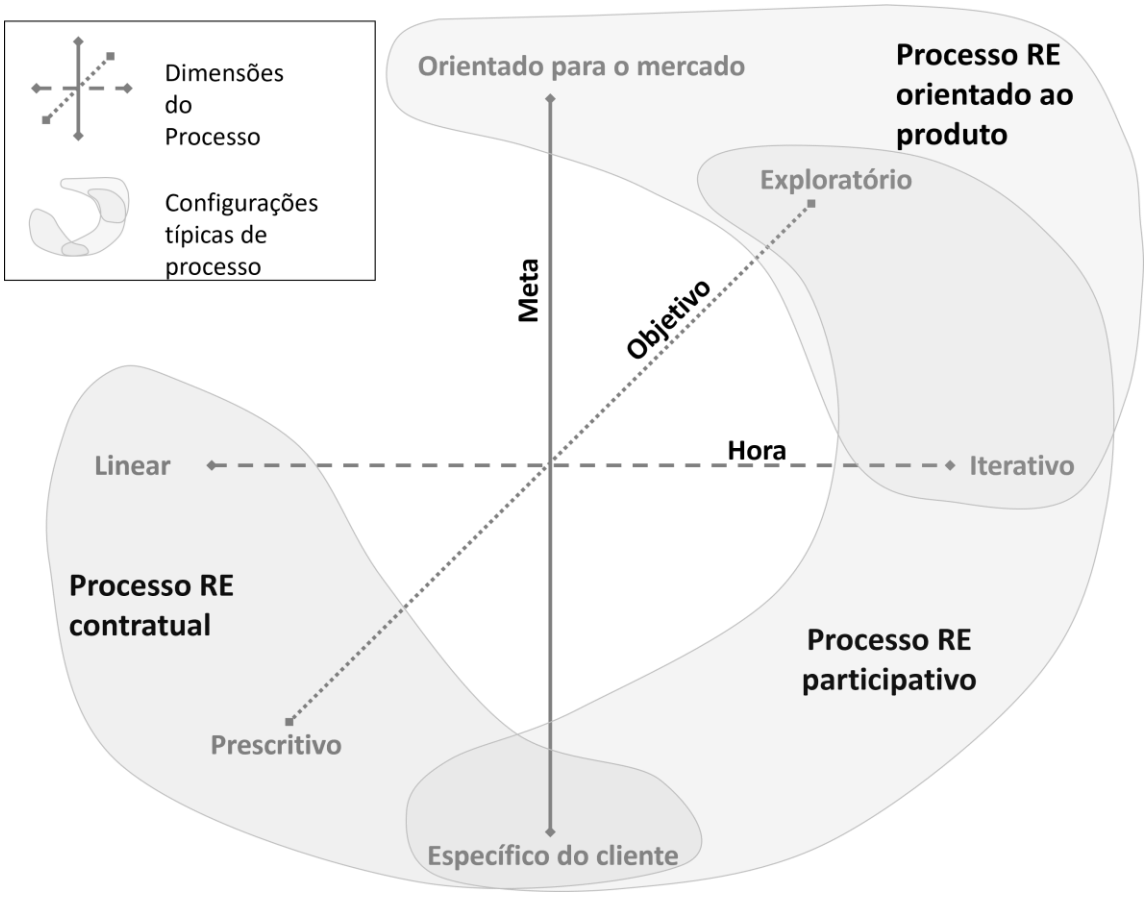


Figura 5.2 Três configurações típicas de processos de ER e sua relação com as três dimensões

Exemplo: Em uma companhia de seguros, a unidade de negócios que vende seguros corporativos para pequenas e médias empresas tem uma ideia sobre um novo produto para segurar os clientes contra os danos sofridos por um ataque de hackers. Eles contratam a unidade de TI corporativa da empresa para formar uma equipe de desenvolvimento com a tarefa de projetar e desenvolver uma nova aplicação que possa lidar com o novo produto de seguros dentro do sistema de suporte de vendas de seguros existente. Além disso, o sistema de gestão de contratos de seguro existente também precisa ser adaptado. Além de alguns requisitos iniciais, a unidade de negócios contratante não tem ideia clara de como o novo produto deve ser e como ele deve ser suportado pelos sistemas de TI corporativos. A TI corporativa adotou o desenvolvimento ágil para todos os seus projetos alguns anos atrás.

Nesta situação, um processo de ER participativo é apropriado. Ela se encaixa no processo geral ágil que a TI corporativa empregará para desenvolver o novo sistema e adaptar os sistemas existentes. Os stakeholders da unidade de negócios e os Engenheiros de Requisitos da TI corporativa podem, em conjunto, levantar os requisitos para o novo produto de seguro. Como o processo é iterativo, a equipe de desenvolvimento pode desenvolver um protótipo de um produto mínimo de mercado (MMP) que ajude a gerência da unidade de negócios a decidir se inclui ou não o produto previsto em seu portfólio ou se descarta a ideia. Existe uma clara relação cliente-fornecedor entre a unidade de negócios e a TI corporativa, portanto, um processo de ER orientado para o cliente se encaixa.

Processo de ER Contratual: Tipicamente Sequencial, Prescritivo e Específico para o Cliente

Um processo de ER Contratual é normalmente escolhido quando o desenvolvimento de um sistema é licitado e terceirizado para um fornecedor com um contrato baseado em uma especificação de requisitos abrangente. É também um processo adequado para a ER em grandes projetos de desenvolvimento de sistemas que aplicam um processo de desenvolvimento ao estilo cascata.

Aplicação principal:	A especificação de requisitos constitui a base contratual para o desenvolvimento de um sistema por pessoas não envolvidas na especificação e com pouca interação dos stakeholders após a fase de requisitos.
Produtos de trabalho típicos:	Especificação clássica dos requisitos do sistema, que consiste em requisitos em linguagem natural e modelados (diagramas)
Fluxo de informação típico:	Principalmente dos stakeholders para os Engenheiros de Requisitos

Exemplo: Um fabricante de automóveis está desenvolvendo uma nova plataforma de carros, da qual uma família de modelos de carros será derivada. Uma grande decisão de projeto para a nova plataforma é se livrar das dezenas de unidades de controle eletrônico (UCEs) atualmente usadas nos carros e substituí-las por um único computador de controle que executa o conjunto de aplicações de controle e assistência à direção. O objetivo é economizar custos de hardware, livrar-se de interações indesejadas entre UCEs e reduzir tanto o tempo quanto o esforço para realizar atualizações do software. Os engenheiros que são responsáveis pelos sistemas eletrônicos da nova plataforma escreveram uma especificação dos requisitos do cliente. A empresa contratou um grande fabricante de sistemas de controle automotivo para criar uma especificação de requisitos de sistema para o novo sistema centralizado de controle de automóveis. Mais tarde, o fabricante de automóveis licitará o projeto e a implementação do sistema com base nessa especificação. O fabricante exigirá que a implementação seja realizada em várias iterações a fim de facilitar os testes e a integração do sistema com a nova plataforma do carro.

Nesta situação, um processo de ER contratual é apropriado. O processo geral é sequencial: o sistema será projetado e implementado somente após a especificação dos requisitos ter sido concluída. O fato de que a implementação será iterativa não afeta o processo de ER. Dependendo da qualidade da especificação dos requisitos do cliente existente e da disponibilidade dos stakeholders no fabricante do carro, deve ser escolhido um processo de ER sequencial ou um processo de ER iterativo.

Obviamente, é necessário um processo de ER Orientado para o cliente. A existência de uma especificação de requisitos do cliente e o fato de que a especificação de requisitos do sistema será usada para licitar o projeto e a implementação do sistema exige um processo de ER prescritivo.

Processo de ER Orientado para o Produto: Iterativo, Exploratório e Orientado para o Mercado

Um processo de ER Orientado ao Produto é normalmente escolhido quando uma organização está desenvolvendo um sistema como um produto ou serviço para o mercado. Na maioria dos casos, um processo de ER orientado ao produto vem junto com um processo ágil de desenvolvimento de produtos. O Product Owners e os designers digitais desempenham papéis importantes neste processo: eles influenciam e moldam fortemente o produto.

Aplicação principal:	Uma organização especifica e desenvolve software para vendê-lo ou distribuí-lo como um produto ou serviço
Produtos de trabalho típicos:	Backlog do produto com histórias de usuários e/ou descrições de tarefas, protótipos
Fluxo de informação típico:	Interação contínua entre stakeholders, Product Owners, Engenheiros de Requisitos e desenvolvedores

Exemplo: Uma empresa de mídia aciona sua TI interna para uma renovação total do aplicativo móvel de notícias que a empresa vende aos assinantes (com algum conteúdo de livre acesso). A partir do feedback dos usuários, a empresa mantém um longo histórico de críticas e sugestões de melhorias.

Em particular, muitos usuários criticam o aplicativo existente por não ser suficientemente responsivo, por ter mau suporte para relatar problemas e sugestões, e por não suportar o zoom com dois dedos de texto ou imagens. O departamento de marketing da empresa também percebe que o layout do aplicativo está desatualizado. Eles preveem que, com um novo layout, mais assinantes poderão ser conquistados. O CEO da empresa decidiu que o departamento de TI deverá colaborar com uma agência externa de design para a aparência visual do aplicativo. A direção da empresa de mídia quer uma versão mínima do produto como prova de conceito, e depois novas versões intermediárias a cada três semanas que possam ser revisadas pelo departamento de marketing e pelo conselho de administração da empresa.

Nesta situação, um processo de ER Orientado ao Produto se encaixa melhor. Embora exista uma relação cliente-fornecedor entre a administração da empresa e seu departamento interno de TI, o foco está claramente na criação de um produto renovado no segmento de aplicações móveis de notícias. O processo de ER precisa ser exploratório, já que não há requisitos claros além do histórico de críticas e sugestões dos usuários. O processo de desenvolvimento deve ser iterativo de acordo com a decisão da gerência da empresa. Como os requisitos precisam ser explorados, um processo de ER Iterativo é o mais adequado aqui.

5.3.2 Outros processos de ER

As três combinações descritas acima cobrem muitas das situações que ocorrem na prática. Entretanto, pode haver situações em que nenhuma das configurações de processo acima mencionadas se encaixe. Por exemplo, restrições regulatórias podem impor a utilização de um processo que esteja em conformidade com determinadas normas, como a ISO/IEC/IEEE 29148 [ISO29148]. Neste caso, o processo de ER tem que ser criado por especialistas de processo a partir do zero ou uma das configurações acima mencionadas tem que ser adaptada para que seja adaptada à situação em questão.

5.3.3 Como configurar Processos de ER

Recomendamos a utilização de um procedimento em cinco etapas ao configurar um processo de ER.

1. *Analisar os fatores de influência.* Analise sua situação com relação à lista de fatores de influência da Seção 5.1.
2. *Avaliar os Aspectos das Dimensões.* Com base na análise da etapa 1, passe pela lista de aspectos de seleção de dimensões apresentada na Seção 5.2. Você pode atribuir a cada critério um valor em uma escala de cinco pontos (––, –, 0, +, ++).

3. *Configurar*. Se a análise dos aspectos produzir um resultado claro com relação às três configurações típicas mencionadas acima, escolha essa configuração. Caso contrário, escolha um processo diferente de adaptação, guiado pelo objetivo geral de mitigar o risco de desenvolver o sistema errado. Por exemplo, imagine uma situação em que o cliente exige que seja criada antecipadamente uma especificação dos requisitos do sistema, o que exige um processo de ER sequencial e prescritivo. Entretanto, em suas primeiras reuniões com o cliente, você notou que, para um subsistema importante, o cliente não tem uma ideia clara do que construir, o que exige um processo de ER exploratório. Uma solução potencial poderia ser escolher um processo de ER contratual como espinha dorsal do processo de ER, mas criar um subprojeto que, por etapas, elicite os requisitos para esse importante subsistema, criando protótipos em duas ou três iterações (guiados por um subprocesso de ER Participativo), e então alimentar os resultados na especificação dos requisitos do sistema.
4. *Determinar produtos de trabalho*. Com base em sua análise e configuração de processo, defina os principais produtos de trabalho de ER que serão produzidos. Certifique-se de que os produtos de trabalho de ER estejam alinhados com os produtos de trabalho do processo de desenvolvimento.
5. *Selecionar as práticas apropriadas*. Para as tarefas a serem realizadas – por exemplo, a elicitação de requisitos – selecione as práticas que melhor se encaixam na situação em questão. Muitas destas práticas, incluindo dicas sobre onde e quando aplicá-las, são apresentadas nos Capítulos 2, 4 e 6 deste guia de estudo.

Não existe um único processo comprovado de ER que se encaixe bem sempre. Com base em uma análise dos fatores de influência, um processo de ER específico precisa ser adaptado para cada empreendimento de ER. Uma forma simples de customizá-lo é configurar um processo de ER a partir de um conjunto de dimensões de processo.

5.4 Leitura adicional

Armour [Armo2004] e Reinertsen [Rein1997], [Rein2009] fornecem reflexões gerais sobre processos e fluxos de informação em processos.

Embora o livro didático de Robertson e Robertson [RoRo2012] se intitule "Mastering the Requirements Process", este é um livro didático geral sobre todos os aspectos da ER.

Wiegers e Beatty [WiBe2013] fornecem um capítulo sobre a melhoria dos processos de ER. O livro de Sommerville e Sawyer [SoSa1998] contém uma coleção de boas práticas a serem utilizadas no âmbito dos processos de ER.

6 Práticas de Gerenciamento de Requisitos

Os requisitos não são esculpidos em pedra, eternamente presentes do passado para o futuro; eles estão vivos! Eles nascem através da elicitação, crescem através da documentação, e são aprimorados através da validação. Como adultos, eles vão trabalhar através da implementação e depois de, oxalá, uma vida longa e próspera em operação, eles se aposentam no esquecimento. Ao longo de seu ciclo de vida, seus pais, os Engenheiros de Requisitos, cuidam deles. Nós cuidamos deles em sua infância, os ensinamos em sua juventude, os acompanhamos em seus relacionamentos e os ajudamos a encontrar um bom emprego em um sistema saudável. A isto chamamos de gerenciamento de requisitos.

Naturalmente, existem definições melhores, mais formais, de gerenciamento de requisitos. A norma ISO/IEC/IEEE 29148:2018 [ISO29148] define o gerenciamento de requisitos como *"atividades que identificam, documentam, mantêm, comunicam, rastreiam e acompanham os requisitos durante todo o ciclo de vida de um sistema, produto ou serviço.*". No glossário CPRE [Glin2020], o gerenciamento de requisitos é definido como "*O processo de gerenciar os requisitos existentes e seus produtos de trabalho relacionados, incluindo o armazenamento, mudança e rastreamento dos requisitos.*". O glossário CPRE também nos diz que o *gerenciamento de requisitos* é parte integrante da Engenharia de Requisitos: "*A abordagem sistemática e disciplinada para especificação e gestão de requisitos com o objetivo de ...".*

O gerenciamento de requisitos pode ocorrer em diferentes níveis:

- Nos requisitos individuais
- Nos produtos de trabalho que contêm estes requisitos
- No sistema relacionado com os produtos de trabalho e os requisitos neles contidos

Na prática, a gestão de requisitos é realizada principalmente no nível do produto do trabalho. Normalmente um produto de trabalho contém vários requisitos individuais (por exemplo, uma descrição de interface externa), enquanto outros produtos de trabalho contêm apenas um único requisito (por exemplo, uma única História de Usuário em um projeto ágil) ou representam todo o conjunto de requisitos para um sistema (por exemplo, uma especificação de requisitos de software). Esteja ciente de que todos os produtos de trabalho dos três níveis devem ser gerenciados, e certifique-se de que você conheça as relações entre eles.

O texto acima delineia o *que é* o gerenciamento de requisitos. O restante deste capítulo é dedicado ao *como*: todos os tipos de práticas que são aplicáveis para fazer funcionar o gerenciamento de requisitos. Antes de mergulharmos nos detalhes do gerenciamento de requisitos, vamos considerar alguns princípios básicos para que possa funcionar a contento. Se você quiser administrar algo, deve ser capaz de reconhecê-lo, armazená-lo e encontrá-lo novamente. Portanto, a identificação única, um grau apropriado de padronização, a prevenção de redundância, um repositório central e o acesso gerenciado são uma necessidade.

Na Seção 6.1, damos uma breve olhada em situações que influenciam o valor, a importância e o esforço envolvidos no gerenciamento de requisitos.

A Seção 6.2 segue os requisitos em seu ciclo de vida como parte dos produtos de trabalho que os Engenheiros de Requisitos e outro pessoal de TI produzem e utilizam ao desenvolver, implementar e operar um sistema de TI.

Durante o ciclo de vida de um requisito, são criadas múltiplas versões de produtos de trabalho (contendo requisitos), começando com um esboço inicial versão 0.1 que, após uma série de grandes e/ou pequenas mudanças, evolui para, digamos, uma versão final 3.2. O controle de versão é discutido na Seção 6.3.

Ao desenvolver e utilizar sistemas de TI, não é conveniente lidar com todos os requisitos de forma individual. Portanto, conjuntos coerentes de requisitos são reconhecidos como configurações, baselines, releases, como explicado na seção 6.4.

A fim de tratar os produtos de trabalho e os requisitos de forma eficiente, devemos ser capazes de identificá-los e coletar dados sobre eles. Esse é o tópico da Seção 6.5.

A Seção 6.6 examina a rastreabilidade dos requisitos. A rastreabilidade é uma característica de qualidade especialmente importante dos requisitos, como você já deve ter entendido ao ler as definições de gerenciamento de requisitos acima. Sem rastreabilidade, é impossível vincular o comportamento real de um sistema aos requisitos originais dos stakeholders.

A Seção 6.7 trata das mudanças nos requisitos que ocorrem durante sua vida útil. Nas primeiras fases de sua existência, as mudanças podem ser frequentes, mas após a validação, os requisitos deveriam estar estáveis. Entretanto, mudanças. irão ocorrer. Para aplicá-las de forma ordenada, deve haver um processo definido para lidar com as mudanças.

Por natureza, os requisitos diferem em importância e valor. Normalmente, os recursos para elaborá-los são limitados, portanto nem todos os requisitos chegarão à implementação. Isto significa que os stakeholders terão que decidir quando um determinado requisito será implementado ou mesmo se ele será implementado ou não. A priorização, descrita na Seção 6.8, pode sustentar esta decisão.

6.1 O que é o Gerenciamento de Requisitos?

Na introdução, já vimos que o gerenciamento de requisitos significa a gestão dos requisitos existentes e dos produtos de trabalho relacionados aos requisitos, incluindo o armazenamento, a mudança e o rastreamento dos requisitos. Mas por que gerenciá-los?

Gerenciamos os requisitos porque eles são seres vivos; eles são criados, usados, atualizados e eventualmente apagados tanto durante seu desenvolvimento quanto sua operação. E durante todo este ciclo de vida, devemos assegurar que todos os stakeholders tenham acesso às versões corretas de todos os requisitos que são relevantes para eles. Se não gerenciarmos os requisitos adequadamente, enfrentamos o risco de que algumas partes possam negligenciar os requisitos, se ater a requisitos ultrapassados, trabalhar com versões erradas, negligenciar relacionamentos entre requisitos, e assim por diante. Isto pode prejudicar seriamente a eficiência e a eficácia do desenvolvimento e do uso do sistema. Em

outras palavras: o valor de uma gestão adequada dos requisitos está na melhoria da eficiência e eficácia de um sistema.

Isto significa que o valor do gerenciamento de requisitos não pode ser separado do valor do sistema em questão e de seu contexto. Na prática, podemos ver enormes diferenças na importância, no nível de gerenciamento de requisitos e no esforço envolvido [Rupp2014], desde uma tarefa eventual e informal de um Engenheiro de Requisitos com uma planilha de cálculo, até uma atividade dedicada e em tempo integral de um gerente de requisitos com um banco de dados de requisitos suportado por ferramentas.

O gerenciamento dos requisitos deve ser mais minucioso quando houver um número maior de requisitos, de stakeholders e de desenvolvedores, quando se espera uma vida útil mais longa do sistema, mais mudanças, maiores demandas de qualidade para o sistema, um processo de desenvolvimento mais complexo for empregado e ter que seguir padrões, normas e regulamentos mais rigorosos, incluindo a necessidade de uma trilha de auditoria detalhada.

Com frequência, vemos que o gerenciamento de requisitos é um tanto negligenciado no início de um projeto, quando uma pequena equipe está trabalhando em um conjunto óbvio de requisitos de alto nível. Mais tarde, a complexidade aumenta e a equipe perde a visão geral, resultando em problemas de qualidade e redução da eficiência. Então, muito esforço tem que ser gasto para alcançar o nível de controle necessário. É mais eficiente investir algum esforço desde o início de um projeto para estabelecer os recursos e processos de gerenciamento de requisitos tendo em mente as demandas esperadas no final.

6.2 Gerenciamento do Ciclo de Vida

Como foi dito na introdução, os requisitos e os produtos de trabalho que contêm requisitos têm vida própria. Vemo-los sendo criados, elaborados, validados, consolidados, implementados, utilizados, alterados, mantidos, retrabalhados, refatorados, retirados, arquivados e/ou apagados. É isso que queremos dizer com seu ciclo de vida: durante sua vida, um requisito pode estar em um número limitado de estados e pode mostrar um número limitado de transições de estados com base em eventos explícitos no contexto. Figura 6.1 mostra um *gráfico de estado* simplificado para o ciclo de vida de um único requisito (apenas uma visão geral, as transições de estado não são mostradas; por exemplo, a transição do estado composto *Em desenvolvimento* para *Em produção* pode ser acionada por uma decisão do Product Owner).

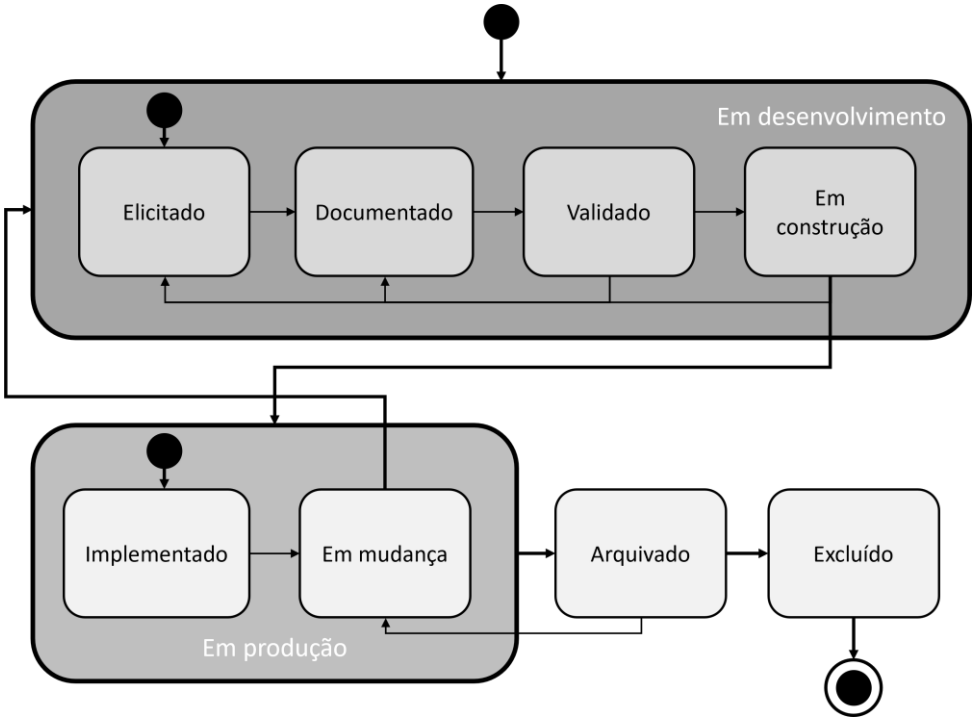


Figura 6.1 Diagrama de estados simplificado do ciclo de vida de um requisito

Um fator complicador é que os produtos de trabalho e os requisitos individuais têm seus próprios e diferentes ciclos de vida que se sobrepõem apenas parcialmente. Como exemplo, pense em um produto de trabalho com o nome *estudos e definições* que se encontra no estado *em mudança*; isto não significa necessariamente que todos os requisitos nele contidos tenham que ser alterados. E para o mesmo *estudos e definições*, o estado *implementado* não faz sentido; apenas alguns requisitos nele serão implementados – ou melhor: determinado código, baseado nestes requisitos.

Outro fator complicador pode ser que, na prática, a visão do ciclo de vida dos requisitos é diferente para diferentes papéis. Para você como Engenheiro de Requisitos, para acompanhar seu trabalho você está interessado em estados diferentes dos do gerente de projeto, do gerente de produto ou do gerente de mudanças: no diagrama acima, seu interesse pode terminar em *validado*, enquanto para o gerente de projeto, ele só começa em *documentado*.

Os Engenheiros de Requisitos gerenciam ativamente o ciclo de vida de seus produtos de trabalho. O gerenciamento do ciclo de vida implica:

- Definir modelos de ciclo de vida para seus produtos de trabalho e os requisitos neles contidos com
- Os estados que um produto de trabalho ou requisito pode assumir
- As transições permitidas entre estes estados
- Os eventos que desencadeiam a transição de um estado para outro
- Assegurar que somente transições explicitamente permitidas possam ocorrer
- Registrar os efetivos estados que os produtos de trabalho e requisitos assumiram
- Registrar as efetivas transições que ocorrem das transições reais que ocorrem

- Reportar sobre estes estados e transições

Em palavras simples: certifique-se de conhecer o estado em que seus requisitos estavam, estão e vão estar, como eles podem mudar, e por que tudo isso acontece.

Por exemplo, como Engenheiro de Requisitos, você poderia ser solicitado a informar quem aprovou qual versão de um requisito liberado para codificação. O acompanhamento dos estados dos requisitos em seu ciclo de vida também pode ser útil para a construção de dashboards e relatórios sobre o progresso de um projeto. Pode ser uma boa maneira de organizar o trabalho e identificar quais os requisitos a serem trabalhados prioritariamente.

O estado de um produto de trabalho sob gerenciamento do ciclo de vida é frequentemente registrado em um atributo (ver Seção 6.5). Também pode ser útil documentar o início e a data final desse estado em atributos. Em projetos ágeis, o estado de um produto de trabalho (item) pode ser derivado de sua posição no backlog de produto, da iteração e/ou das tarefas e/ou no quadro de tarefas. Além disso, o cumprimento dos critérios da *definição de preparado* (DoR) e da *definição de feito* (DoD) pode dar informações relevantes, pois o cumprimento desses critérios significa, na verdade, atingir um próximo estado.

O rigor e o nível de detalhamento do gerenciamento do ciclo de vida devem ser adaptados às necessidades do cliente, do projeto e do sistema. Por exemplo, os estados em desenvolvimento, em produção e descontinuado podem ser suficientes. Em projetos complexos ou críticos, você pode precisar de um modelo muito mais detalhado dos estados, procedimentos rigorosos sobre transições de estado e uma trilha de auditoria que mostre o que aconteceu durante o projeto.

6.3 Controle de Versão

É comum que tanto produtos de trabalho quanto seus requisitos individuais sofram certas mudanças durante seu ciclo de vida (consulte a Seção 6.7 para obter mais informações sobre como lidar com essas mudanças). Após cada mudança, o produto de trabalho é diferente do que era antes: ele se tornou uma nova versão.

Queremos controlar as versões destes produtos de trabalho por duas razões:

- Às vezes as mudanças dão errado. Após algum tempo, defeitos são encontrados, ou os benefícios pretendidos não são alcançados. Nesse caso, podemos implementar novas mudanças em uma próxima versão, mas também podemos decidir voltar a uma versão anterior e continuar a partir dela. Ou talvez, pensando bem, simplesmente preferimos a versão anterior.
- Queremos conhecer a história do produto de trabalho, compreender sua evolução desde sua origem até sua situação atual. Isto pode nos ajudar quando tivermos que decidir sobre mudanças futuras, ou apenas responder a perguntas sobre o porquê do produto de trabalho atual ser o que é.

O controle de versões requer a aplicação de três medidas:

- Uma *identificação* de cada versão, para distinguir entre as diferentes versões de um produto de trabalho. Este é o número da versão, muitas vezes complementado com uma data de versão.
- Uma descrição clara de cada *mudança*. Você deve ser capaz de dizer – e compreender – a diferença entre uma determinada versão e sua predecessora. Esta descrição de mudança deve estar claramente vinculada ao número da versão.
- Uma política rigorosa sobre o *armazenamento* de versões, permitindo localizar e recuperar versões antigas. A menos que as limitações de armazenamento determinem o contrário, você deve preservar todas as versões anteriores de todos os seus produtos de trabalho, caso contrário, você pode não ser capaz de restaurar uma versão se precisar dela. Por outro lado, o armazenamento ilimitado raramente será o caso, portanto é sábio ter também uma política de arquivamento e remoção de produtos de trabalho que não são mais utilizados.

Normalmente, um produto de trabalho contém múltiplos requisitos. Se um único requisito nesse produto de trabalho mudar, tanto esse requisito quanto o produto de trabalho devem obter um novo número de versão, enquanto os requisitos inalterados nesse produto de trabalho mantêm seu número de versão antigo. Isto pode logo se tornar muito confuso. Uma solução prática pode ser fazer a numeração de versão somente no nível do produto de trabalho e deixar que todos os requisitos nele contidos herdem o número da versão e o histórico de mudanças do produto de trabalho.

Os números das versões são normalmente compostos de (pelo menos) duas partes:

- *Versão*. Em princípio, a versão começa no *zero* enquanto o produto de trabalho estiver em desenvolvimento. Quando é formalmente aprovado, liberado e/ou lançado, lhe atribuímos a versão *um*. Depois disso, a versão é aumentada apenas quando de atualizações substantivas.
- *Incremento*. Isto geralmente começa em *um* e é incrementado com cada mudança (externamente visível) de conteúdo ou apenas por mudança textual ou editorial. Um subincremento adicional pode ser usado apenas para correção de erros de digitação. O incremento *nove* é às vezes usado para denotar uma versão final pouco antes da aprovação ou liberação.

Um novo número de versão é atribuído a cada mudança formal.

Muitas vezes, uma mudança no estado do ciclo de vida de um produto de trabalho não é considerada um motivo para aumentar o número da versão, a menos que seja acompanhada por uma mudança no conteúdo ou texto. Se, por exemplo, um requisito recebe o estado validado e o número de versão 1.0 após a aprovação, não há necessidade de alterar este número de versão se o estado mudar para em construção e, posteriormente, para implementado. O estado pode finalmente terminar em indisponível, mas ainda assim manter a mesma versão número 1.0.

6.4 Configurações e Baselines

Suponha que você preserve, como aconselhado acima, todas as versões de todos os requisitos que você desenvolve durante um projeto. Você terá então um banco de dados sempre em expansão, repleto de requisitos, e começará a perder a visão geral. Um dia, seu cliente vêm até sua mesa e pergunta: "Nós implementamos seu sistema em todas as nossas filiais. Agora parece haver um problema com os cálculos em nosso escritório de Barcelona. Você pode me dizer qual versão dos requisitos de cálculo eles usam lá"? Se você não puder responder a essa pergunta, desejará ter prestado mais atenção ao gerenciamento de configurações.

Então, o que é uma configuração? Você encontrará uma definição no glossário CPRE [Glin2020] mas em resumo, para um Engenheiro de Requisitos, uma configuração é um conjunto consistente de produtos de trabalho logicamente relacionados que contêm requisitos. Selecionamos este conjunto com um objetivo específico, geralmente para deixar claro quais requisitos são ou foram válidos em uma determinada situação.

As seguintes propriedades definem uma configuração correta:

- *Logicamente conectada*. O conjunto de requisitos da configuração está unido em vista de um certo objetivo.
- *Consistente*. O conjunto de requisitos não tem conflitos internos e pode ser integrado em um sistema.
- *Único*. Tanto a configuração em si como seus requisitos constituintes são identificados de forma clara e única.
- *Imutável*. A configuração é composta de requisitos selecionados, cada um com uma versão específica que nunca será alterada nesta configuração.
- *Base para Restauração*. A configuração permite o fallback (retorno, restauração) para uma configuração anterior se quaisquer efeitos indesejados tenham ocorrido.

Uma configuração é documentada como um produto de trabalho, com uma identificação única, um estado, um número de versão e uma data, como qualquer outro produto de trabalho. Entretanto, como uma configuração é, por definição, imutável, ela sempre terá apenas uma versão (por exemplo, 1.0).

Uma configuração sempre tem duas dimensões [CoWe1998]:

- A dimensão *do produto*.

  Isto indica quais requisitos estão incluídos nesta configuração específica. Às vezes, uma configuração contém todos os requisitos disponíveis, mas geralmente é uma certa seleção – por exemplo, todos os requisitos que são implementados no lançamento francês de um sistema. O lançamento britânico do mesmo sistema pode então ter uma configuração diferente.

- A dimensão da *versão*.

  Em uma configuração específica, cada requisito selecionado está presente em exatamente uma, e apenas uma, versão. Pode ser a última versão ou uma anterior,

dependendo da finalidade da própria configuração. Tão logo até mesmo uma única versão diferente de um único requisito for selecionada, esta será uma nova configuração. Imagine um sistema para o qual um novo lançamento será implementado com alguns requisitos em uma versão superior: este novo lançamento terá então uma configuração diferente.

Figura 6.2 dá outro exemplo de configurações diferentes que consistem em conjuntos específicos de versões de requisitos.

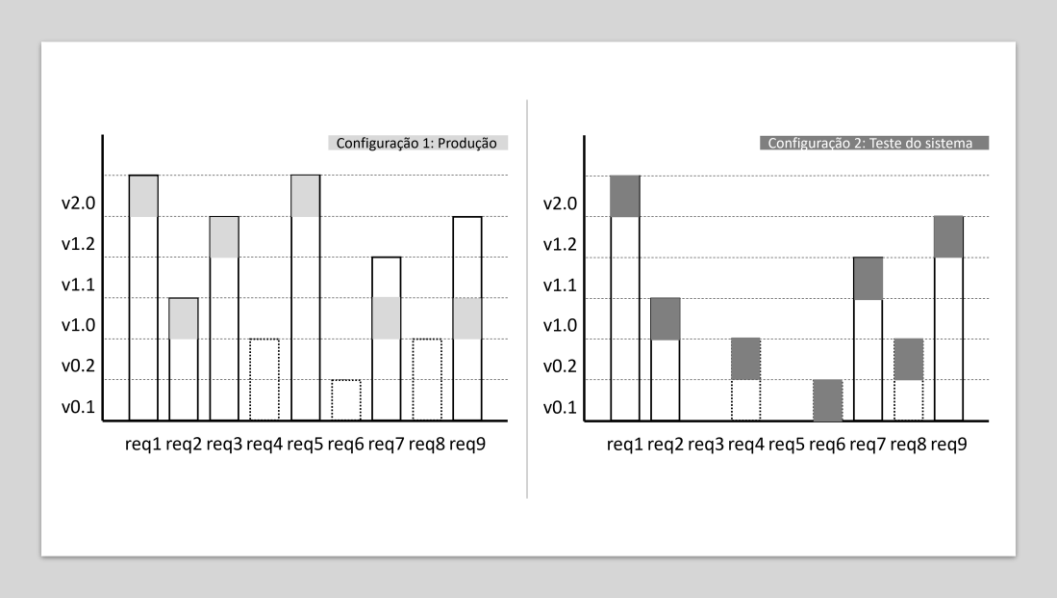


Figura 6.2 Exemplo de configurações

A figura acima mostra um exemplo de diferentes configurações de um determinado sistema. Ele mostra uma coleção de nove requisitos. Alguns deles ainda estão nos estágios iniciais de desenvolvimento – por exemplo, o requisito 6 com a versão v0.1. Outros requisitos tiveram mais versões – por exemplo, o requisito 1, que está finalizado e já sofreu uma grande atualização, por isso agora está na versão v2.0.
A imagem à direita mostra a configuração que está atualmente em produção. Ele consiste em R1 v2.0, R2 v1.0, R3 v1.2 (este requisito teve duas pequenas atualizações após a implementação), R5 v2.0, R7 v1.0, e R9 v1.0. R4, R6 e R8, estando em desenvolvimento, não estão presentes nesta configuração, nem as novas versões de R7 e R9.
A imagem à esquerda mostra a configuração que, ao mesmo tempo, está presente no ambiente de teste do sistema. Alguns requisitos (R1, R2) são os mesmos, alguns não estão mais presentes (R3, R5), os requisitos em desenvolvimento (R4, R6 e R8) estão incluídos aqui, e dois requisitos (R7 e R9) estão presentes em uma versão mais alta do que na configuração do ambiente de produção.

Em muitos projetos algumas configurações são tratadas de uma maneira especial: estas configurações são chamadas de baseline. Uma *baseline* é uma configuração estável, validada e sob gestão controlada de mudanças que sinaliza um marco ou outro *ponto de avaliação* no projeto. Um exemplo pode ser a configuração no final da fase de projeto, antes

de iniciar a fase de codificação, ou a configuração que é válida na liberação em produção de uma determinada release.

O backlog da sprint em um projeto ágil serve como baseline no início da próxima iteração. As baselines são úteis para fins de planejamento, pois representam um ponto de partida estável para uma próxima fase. Eles são frequentemente congelados e colocados de lado como âncora na vida agitada de um projeto. Se algo correr terrivelmente mal no projeto, a equipe pode fazer um rollback para a situação da baseline e reiniciar a partir dela.

Para o Engenheiro de Requisitos, é principalmente a configuração de produtos de trabalho que contém requisitos que é importante. Mas, na prática, a configuração dentro de um projeto tem um escopo muito mais amplo, contendo versões selecionadas dos produtos de trabalho de todos os membros da equipe, tais como requisitos, desenhos, códigos e casos de teste. Em projetos complexos, o gerenciamento da configuração pode ser um trabalho em tempo integral, realizado com ferramentas dedicadas.

6.5 Atributos e Visualizações

Como Engenheiro de Requisitos, sua produção consiste em todos os tipos de produtos de trabalho que contêm requisitos. Estes requisitos terão que ser gerenciados, caso contrário, você e sua equipe rapidamente perderão a visão geral. Para gerenciar os requisitos, é preciso coletar e manter dados sobre eles – metadados, dados sobre dados. Metadados tornam os produtos de trabalho tangíveis, gerenciáveis; através de metadados, você pode fornecer e obter informações sobre os requisitos e responder perguntas que são relevantes durante e após o projeto ou ciclo de vida do produto. Pense em perguntas como *"Quais requisitos estão planejados para o próximo lançamento?*" ou *"Quanto esforço esta liberação provavelmente exigirá?*" ou "*Quantos requisitos têm alta prioridade?"*

Ao considerar os requisitos como entidades sobre quais informações são requeridas, as características destes requisitos são chamadas *atributos*. Neste capítulo, já vimos alguns atributos comuns, tais como a identificação única, número da versão, estado, várias datas. Os atributos a serem definidos para os requisitos dependem das necessidades de informação dos stakeholders do projeto e do sistema. No início de um projeto, deve ser definido um *esquema de atributos* que permita ao Engenheiro de Requisitos atender a essas necessidades.

Um bom ponto de partida pode ser encontrado nas normas relevantes. A norma ISO [ISO29148] menciona:

- *Identificação*. Cada requisitos deve ter um identificador único e imutável, como um número, nome, mnemônico. Sem uma identificação adequada, a gestão de requisitos é impossível.
- *Prioridade para os stakeholders.* A prioridade (acordada) do requisito do ponto de vista dos stakeholders. Consulte a Seção 6.8 para informações sobre como determinar esta prioridade.

- *Dependências.* Às vezes, há dependências entre requisitos. Isto pode significar que um requisito de baixa prioridade deve ser implementado primeiro porque outro requisito de alta prioridade depende disso.
- *Risco*. Trata-se do potencial da implementação do requisito levar a problemas, tais como danos, custos extras, atrasos, ações judiciais. Por natureza, esta é uma estimativa, a ser obtida pelo consenso entre os stakeholders.
- *Fonte*. Qual é a origem do requisito, de onde veio? Você pode precisar dessas informações para validação, resolução de conflitos, modificação ou remoção.
- *Justificativa*. A justificativa elenca a razão do requisito ser necessário, os objetivos dos stakeholders atendidos pela sua implementação.
- *Dificuldade*. Esta é uma estimativa do esforço necessário para implementar o requisito. Ela é necessária para o planejamento e estimativa de projetos.
- *Tipo*. Este atributo indica se o requisito é funcional de qualidade ou de restrição.

Existem muitas maneiras de armazenar estas informações. Podem estar contidas em documentos ou armazenada em uma planilha ou banco de dados, com os requisitos como linhas e seus atributos como colunas. Em ambientes ágeis, os requisitos podem ser registrados em cartões de histórias, onde os atributos são nele anotados. Como discutido no Capítulo 7, as ferramentas de gerenciamento de requisitos devem oferecer funcionalidade para o armazenamento de dados sobre os requisitos e também para a elaboração de relatórios sobre eles.

Os atributos permitem que você forneça informações sobre seus produtos de trabalho e seus requisitos. A maneira mais simples de fazer isso é gerar um relatório com todos os dados sobre todas as versões de todos os requisitos. Exceto para sistemas simples, tal relatório será inútil, pois ninguém será capaz de supervisionar todas as informações pois a complexidade é excessiva. Portanto, você deve ajustar seus relatórios com base nas necessidades de informação de seus públicos-alvo. Isto é feito utilizando *visualizações* [Glin2020].

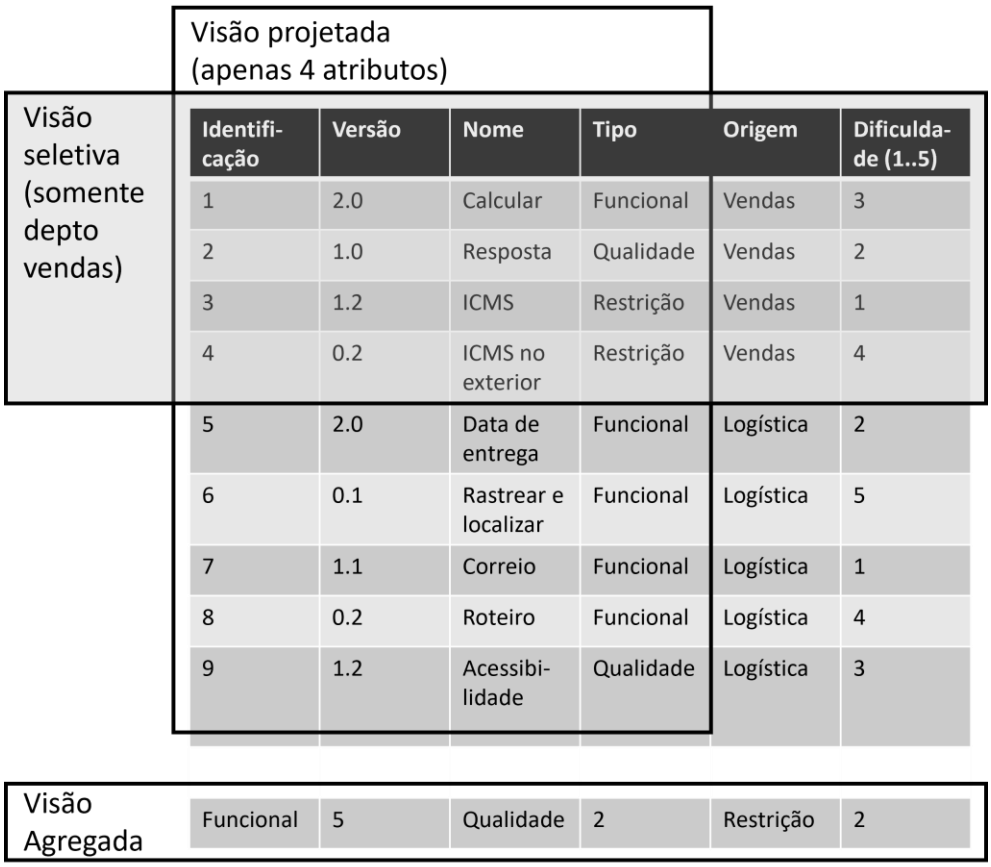
Uma visualização é uma forma (muitas vezes predefinida) de filtrar e classificar os dados sobre seus produtos de trabalho, resultando em um relatório que mostra exatamente o que o público precisa, nem mais, nem menos. Uma visão é definida com o propósito explícito de fornecer informações relevantes para um grupo alvo específico.

Distinguimos três tipos de visualizações:

- *Visualizações Seletivas.* Estas visualizações fornecem informações sobre uma seleção deliberada dos requisitos e não de todos. Por exemplo, uma visualização apenas sobre as últimas versões dos requisitos, ou todos os requisitos com o estado *validado*, ou sobre os requisitos com prioridade *elevada* para os stakeholders; o foco pode ser um subsistema, ou ao contrário, fornecer uma visão abstrata do sistema através de seus requisitos de alto nível apenas.
- *Visualizações Filtradas.* Uma visualização filtrada mostra uma seleção de parte dos atributos dos requisitos – por exemplo, apenas a identificação, o número da versão e o nome.

- *Visualizações Agregadoras*. Em uma visualização agregadoras, você encontrará resumos, totais ou médias, calculados a partir de um conjunto de requisitos. Um exemplo seria o número total de necessidades por departamento: por exemplo, 4 de Vendas, 5 de Logística.

Figura 6.3 dá um exemplo desses tipos de visualizações.



Identifi-cação	Versão	Nome	Tipo	Origem	Dificulda-de (1..5)
1	2.0	Calcular	Funcional	Vendas	3
2	1.0	Resposta	Qualidade	Vendas	2
3	1.2	ICMS	Restrição	Vendas	1
4	0.2	ICMS no exterior	Restrição	Vendas	4
5	2.0	Data de entrega	Funcional	Logística	2
6	0.1	Rastrear e localizar	Funcional	Logística	5
7	1.1	Correio	Funcional	Logística	1
8	0.2	Roteiro	Funcional	Logística	4
9	1.2	Acessibi-lidade	Qualidade	Logística	3
Funcional	5	Qualidade	2	Restrição	2

Figura 6.3 Diferentes tipos de visualizações

Na maioria dos casos, uma combinação de visualizações é usada, por exemplo, se você quiser fornecer uma lista com os IDs, números de versão, nomes e tipos (= visualização filtrada) de todos os requisitos para o departamento de vendas (= visualização seletiva).

6.6 Rastreabilidade

Ao longo deste guia de estudo, mencionamos o tópico de rastreabilidade [GoFi1994]. Sem uma rastreabilidade adequada, a Engenharia de Requisitos é dificilmente viável, pois não se pode fazer o seguinte:

- Fornecer provas de que um determinado requisito é satisfeito
- Comprovar que um requisito foi implementado e por que meios
- Mostrar a conformidade do produto com as leis e normas aplicáveis

- Procurar produtos de trabalho em falta (por exemplo, descobrir se existem casos de teste para todos os requisitos)
- Analisar os efeitos de uma mudança nos requisitos (ver Seção 6.7)

Em muitos casos, especialmente para sistemas onde a segurança é crítica, as normas existentes exigem explicitamente a implementação da rastreabilidade.

Há três tipos de perguntas que podem ser respondidas com a ajuda da rastreabilidade (ver também Figura 6.4):

- *Rastreabilidade retroativa*: Qual foi a origem de um determinado requisito? Onde foi encontrado? Que fontes (stakeholders, documentos, outros sistemas) foram analisadas durante a elicitação?

  A rastreabilidade retroativa também é conhecida como a rastreabilidade pré-especificação de requisitos.

- *Rastreabilidade progressiva*: Onde este requisito é utilizado? Quais os produtos (módulos codificados, casos de teste, procedimentos, manuais) que se baseiam nele?

  A rastreabilidade progressiva também é conhecida como rastreabilidade pós-especificação de requisito.

- *Rastreabilidade entre requisitos*: Que requisitos dependem de outros requisitos ou vice-versa (por exemplo, requisitos de qualidade relacionadas a um requisito funcional)? O requisito é um refinamento de um requisito de nível superior (por exemplo, um refinamento de um épico em várias histórias de usuários, uma História de Usuário detalhada com uma série de critérios de aceite)? Como eles estão relacionados?

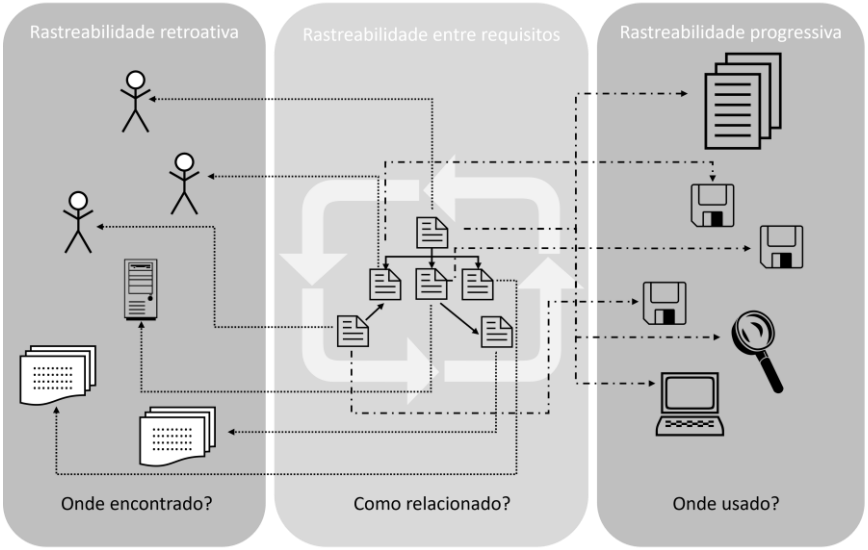


Figura 6.4 Tipos de Rastreabilidade

Há várias maneiras de documentar a rastreabilidade. Muitas vezes, isto é feito *implicitamente,* por exemplo, aplicando estruturas padronizadas de documentos, templates padrão, ou convenções de nomenclatura. Se você identificar todos os seus requisitos com o código *Req-xxx-nnn*, onde *xxx* representa o departamento que solicitou o requisito, todos entenderão que Req-VND-012 é um requisito para o departamento de Vendas (para rastreabilidade retroativa). Se você publicar um documento listando todos os requisitos que serão implementados no lançamento de 1º de julho, você estará fornecendo informações implícitas de rastreabilidade progressiva.

E se você escrever um documento com uma seção dedicada, por exemplo, a cálculos de preços, isso poderia ser um exemplo de rastreabilidade entre requisitos. Outro exemplo poderia ser um modelo de alto nível e uma descrição textual dos requisitos detalhados relacionados a ele.

Em projetos mais complexos, a rastreabilidade deve (também) ser documentada *explicitamente*. Para uma rastreabilidade explícita, você documenta a relação entre os produtos de trabalho com base em sua identificação única. Isto pode ser feito de várias formas [HuJD2011]:

- Fazendo uso de atributos específicos, como *Fonte* sugerida pela norma ISO [ISO29148]
- Em documentos, acrescentando referências a documentos predecessores, outros produtos de trabalho ou requisitos individuais
- Desenvolvimento de uma matriz de rastreabilidade em uma planilha, ou em uma tabela de banco de dados (veja um exemplo em Tabela 6.1 abaixo)
- Na documentação textual, usando hyperlinks no estilo Wiki
- Visualização das relações de rastreabilidade em um gráfico *de rastreamento* (Figura 6.4 é uma forma simplificada de um gráfico desse tipo)

Em muitos casos, uma ferramenta de gerenciamento de requisitos ou de configuração (ver Capítulo 7) fornece funcionalidade para apoiar a rastreabilidade. Gerenciar a rastreabilidade em um projeto substancial pode ser complicado, especialmente se você também tiver que levar em conta o versionamento. Em tal caso, é indispensável uma boa ferramenta.

Tabela 6.1 Exemplo de uma matriz de rastreabilidade

Fonte	R1	R2	R3	R4	R5	R6	R7
Entrevista Sra. Smith 06/08	X	X			X		
Resumo do questionário de 12 de maio	X			X		X	X
Relatório de observação de campo 07/03			X	X	X		
Regulamento da empresa versão 17.a.02			X			X	X
Documentação API sistema HRM v3.0.2.a	X			X	X		

6.7 Lidando com Mudanças

Relembre o princípio 7: Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Os processos ágeis aproveitam as mudanças para a vantagem competitiva do cliente." [BeeA2001]. Os pais fundadores do movimento ágil foram muito claros sobre isto: mudanças de requisitos sempre ocorrerão, quer você goste ou não. Muitas pessoas não gostam nada das mudanças, porque cada mudança é um risco, uma ameaça para a estabilidade do projeto e do sistema.

Entretanto, mudar um requisito não é um evento isolado: é desencadeado por mudanças no contexto do sistema, por novas percepções dos stakeholders, pelo comportamento dos concorrentes etc.; uma lei entra em vigor, acrescentando uma nova restrição ao sistema; devido à crescente demanda do mercado, o desempenho do sistema tem que ser melhorado; um sistema concorrente é lançado com algumas características *mais encantadoras* que seu cliente também deseja. Uma mudança deve, portanto, ser vista como uma oportunidade de obter um sistema melhor, para fornecer mais valor aos usuários.

Entretanto, independentemente da situação, cada mudança é também um risco. Ele pode introduzir defeitos, levando a falhas no sistema. Pode atrasar o progresso do projeto. Pode exigir mais esforço e dinheiro do que foi calculado antes. Os usuários podem não gostar e se recusar a trabalhar com ele. Em resumo, as coisas podem dar errado e perturbar um projeto ou sistema anteriormente estável. Mas isso não significa que as mudanças são ruins e devem ser evitadas; significa, sim, que todas as mudanças devem ser tratadas com cuidado para obter o valor ideal a custos aceitáveis com o mínimo de risco.

Na literatura sobre gerenciamento de serviços de TI (ver [Axelos2019]), a *habilitação da mudança* é descrita como uma das práticas centrais. Esta prática garante que as mudanças sejam implementadas de forma eficaz, segura e oportuna para atender às expectativas dos stakeholders. A prática equilibra eficácia, rendimento, conformidade e controle de risco. Ela se concentra em três aspectos:

- Garantir que todos os riscos tenham sido avaliados com precisão
- Autorizar mudanças para prosseguir
- Gerenciar a implementação da mudança

A habilitação para mudanças implica que uma organização designe uma autoridade de mudança para decidir sobre as mudanças e definir um processo para tratá-las. Veja Figura 6.5 para um esboço deste processo. Estas medidas estão geralmente ajustadas à abordagem de desenvolvimento e ao ponto no tempo em que ocorre uma mudança.

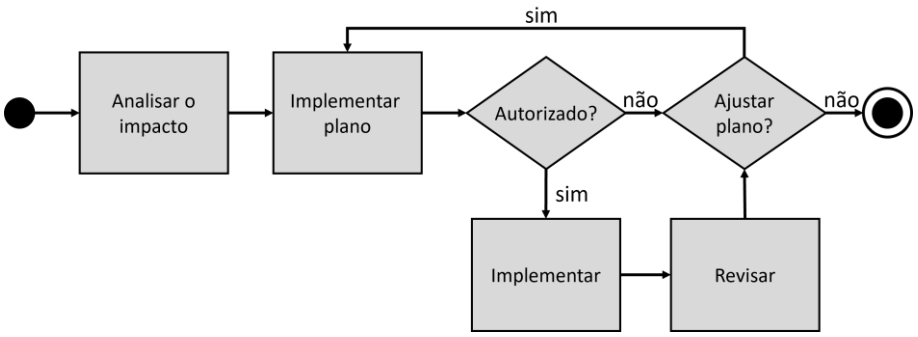


Figura 6.5 Processo de habilitação de mudança

Enquanto um requisito estiver em um estado de rascunho, o autor tem autoridade para modificá-la e nenhum processo rigoroso é seguido.

Assim que um requisito for liberado para uso posterior no projeto, o autor não terá mais liberdade para decidir, pois cada mudança terá um impacto em outros produtos de trabalho nele baseados. Antes de decidir se uma mudança deve ser implementada, uma análise de impacto deve ser realizada para esclarecer os esforços e riscos da mudança. É aqui que a rastreabilidade é indispensável. Em uma abordagem de desenvolvimento *sequencial*, a autoridade de mudança será frequentemente designada para a gestão do projeto, um comitê de direção ou um *Conselho de Controle de Mudanças*, e um processo é seguido, com uma decisão formal sobre a mudança e o planejamento de sua implementação. Em uma abordagem de desenvolvimento *iterativo*, a autoridade de mudança geralmente é o Product Owner, que decide sobre a mudança e acrescenta uma mudança aceita como um novo item (produto de trabalho) no backlog do produto. A implementação posterior é então tratada como qualquer outro item do backlog do produto.

Uma vez que um requisito já esteja em produção em um sistema, um processo ainda mais rigoroso deve ser seguido, pois cada mudança agora influenciará os usuários e os processos do negócio.

Aqui, muitas vezes é feita uma distinção entre *mudanças simples* (baixo risco, bem compreendida e pré-autorizada, por exemplo, uma mudança na porcentagem do ICMS), *mudança normais* (baseado em um Pedido de Mudança formal, programado, avaliado e autorizado, por exemplo, uma mudança em um algoritmo de cálculo de preço), e *mudanças emergências* (a serem implementadas o mais rápido possível, por exemplo, para resolver um incidente – mas que raramente envolve uma mudança de requisitos). Normalmente, a autoridade de mudança é um *Conselho Consultivo de Mudança* [Math2019]; em uma abordagem iterativa como DevOps, uma mudança pode ser autorizada por um gerente de liberação.

6.8 Priorização

OS requisitos por si só são apenas conceitos na mente das pessoas. Eles trazem valor somente quando são implementados em um sistema e postos em operação. Esta implementação requer esforço, tempo, dinheiro e atenção. Na maioria dos casos, esses

recursos são limitados, o que significa que nem todos os requisitos podem ser implementados, pelo menos não ao mesmo tempo. Isto, por sua vez, significa que os stakeholders têm que decidir quais requisitos devem ser implementados primeiro e quais poderiam ser implementados mais tarde (ou não ser implementados). Em outras palavras: priorização [Wieg1999].

A prioridade de um requisito é definida como o nível de importância atribuído a ele de acordo com determinados critérios [Glin2020]. Consequentemente, primeiro é preciso determinar quais critérios devem ser usados para avaliar os requisitos antes de poder priorizá-los. Entretanto, antes de poder determinar os critérios de avaliação, você deve saber qual é o objetivo da priorização. Esse objetivo geralmente não é seu objetivo como Engenheiro de Requisitos, mas o objetivo de certos stakeholders portanto você deve decidir quem são. E quando você conhecer seu objetivo, geralmente ficará claro que nem todos os requisitos terão que ser priorizados, mas apenas um subconjunto definido.

Resumindo o acima exposto, podemos delinear uma sequência de passos a serem seguidos se quisermos priorizar os requisitos:

- Definir as principais metas e restrições para a priorização

  O contexto do projeto e do sistema determinam em grande parte as razões para a priorização. Se, por exemplo, você priorizar para decidir quais características serão implementadas na próxima release, você pode se concentrar no valor negocial; se o objetivo for selecionar histórias de usuários para a próxima iteração, os story points e as dependências técnicas seriam mais proeminentes. Restrições técnicas ou legais podem limitar as escolhas a serem feitas.

- Definir os critérios de avaliação desejados

  Em princípio, os objetivos e restrições ditam os critérios a serem utilizados. Os critérios comumente utilizados são o valor negocial para os stakeholders, a urgência percebida pelos usuários, o esforço para implementar, os riscos de uso, as dependências lógicas e técnicas, a natureza juridicamente vinculativa de um requisito, ou apenas a preferência subjetiva de (ou entre) stakeholders relevantes. Às vezes é utilizado apenas um único critério, mas uma seleção equilibrada de vários critérios relevantes pode produzir um resultado melhor.

- Definir os s*takeholders* que têm de estar envolvidos

  As metas e restrições influenciam quais stakeholders você deve envolver na priorização, mas, por outro lado, certos stakeholders estabelecem eles mesmos estes objetivos, portanto você deve estar ciente desta interdependência. Como exemplo, ao dar prioridade ao lançamento de um novo sistema, você provavelmente convidaria representantes do negócio e um painel de futuros clientes. Ao priorizar o backlog do produto para decidir sobre a próxima iteração, a equipa Scrum deverá estar envolvida.

- Definir os *requisitos* a serem priorizados

É improvável que todo o conjunto de requisitos tenha que ser priorizado. Mais uma vez, isto depende principalmente dos objetivos e restrições para priorizar. Por exemplo, restrições podem exigir que certos requisitos sejam obrigatoriamente implementados. Na verdade, só é útil priorizar os requisitos para os quais você tem a escolha de incluí-los ou não em uma próxima etapa do processo de desenvolvimento. Isto significa que a fase de projeto também é um fator importante. Em uma fase inicial, você pode incluir versões preliminares na priorização; em uma fase tardia, você frequentemente restringirá a priorização aos requisitos que estão em uma versão estável. Esteja ciente de que os requisitos a serem priorizados devem estar em um nível comparável de abstração, dependendo das metas de priorização. Em uma fase inicial do projeto, por exemplo, você pode priorizar épicos, temas ou features enquanto no planejamento da iteração você irá priorizar as histórias de usuários.

- Selecionar a *técnica* de priorização

  Uma técnica de priorização é a forma pela qual os seus Stakeholders priorizam os requisitos. Como descrito abaixo, existem várias técnicas, que diferem em esforço, rigor e detalhamento. Aqui também, objetivos e restrições definem o cenário, mas o fator mais importante é que os stakeholders envolvidos concordem sobre a técnica que eles pretendem utilizar. Caso contrário, eles não aceitarão o resultado e seu esforço de priorização será em vão.

- Realizar priorização

  Quando toda a preparação tiver sido feita, a priorização de facto poderá se realizar. Primeiro, todos os critérios de avaliação definidos devem ser aplicados a todos os requisitos selecionados. Juntamente com os participantes envolvidos, você aplica a técnica de priorização selecionada aos requisitos avaliados. Como resultado, você obtém uma lista priorizada de requisitos. No entanto, pode haver um problema. Os diferentes stakeholders podem ter prioridades diferentes, mesmo se concordarem com os critérios avaliados. Nesse caso, você normalmente tem um conflito de requisitos que deve ser resolvido como qualquer outro conflito, como descrito na Seção 4.3 sobre resolução de conflitos.

Analisando mais de perto as técnicas de priorização, fazemos distinção entre duas categorias:

- Técnicas ad hoc

  Com t*écnicas ad hoc*, os especialistas atribuem prioridades aos requisitos selecionados com base em sua própria experiência. Em princípio, esta priorização se baseia em um único critério, sendo a percepção subjetiva do especialista. Se esta competência for de um nível elevado e aceitável para as partes interessadas, esta técnica pode ser uma forma rápida, barata e fácil de estabelecer prioridades. Uma variante seria convidar vários especialistas e calcular algum tipo de prioridade média. Técnicas ad hoc comuns incluem a classificação Top-10 e a priorização MoSCoW (Must have, Should have, Could have, Won't have this time). A análise Kano (Seção

4.2.1) também é útil: os fatores básicos são obrigatórios, os fatores esperados deveriam ter, e os fatores de entusiasmo poderiam ter ou não ter. Para mais informações, ver, por exemplo, [McIn2016].

- *Técnicas analíticas*

  As *técnicas analíticas* empregam um processo sistemático para a atribuição de prioridades. Nessas técnicas, os especialistas atribuem pesos a múltiplos critérios de avaliação (tais como benefício, custo, risco, tempo para implementar etc.) e, posteriormente, as prioridades dos requisitos são calculadas como resultados ponderados com base nesses critérios. Tais técnicas exigem mais esforço e tempo, mas têm a vantagem de dar uma visão clara dos fatores que determinam as prioridades e do processo pelo qual as prioridades são estabelecidas. Isto pode estimular o aceite do resultado entre os stakeholders. Entretanto, dois aspectos devem ser considerados. Primeiro, o resultado é fortemente influenciado pelos fatores de peso ponderado que são usados no cálculo do resultado. Portanto, um acordo entre os stakeholders sobre esses fatores de peso deve ser estabelecido antes da priorização propriamente dita. Caso contrário, alguns poderiam tentar mudar os fatores de peso a fim de manipular as prioridades. O segundo aspecto a considerar é que os critérios avaliados são principalmente estimativas, não fatos medidos. E as estimativas são frequentemente em uma escala ordinal simples, como baixa, média, alta. Portanto, a qualidade das estimativas é decisiva para a qualidade da priorização resultante. Entretanto, as técnicas analíticas são úteis para proporcionar uma priorização claramente fundamentada que seja compreendida e, portanto, aceita pelos stakeholders envolvidos. Para uma explicação detalhada das técnicas analíticas, ver [Olso2014].

Pode ser tentador aplicar técnicas detalhadas e minuciosas e gastar muito tempo produzindo estimativas perfeitamente precisas em termos de dinheiro, horas, números de vendas esperadas etc. Isto poderia resultar no requisito A ter uma prioridade calculada de 22,76, requisito B de 23,12, e requisito C de 20,29. Você concluiria então que, evidentemente, C deve ser feito primeiro e A antes de B. Entretanto, você provavelmente acabou de introduzir uma pseudo-precisão com este cálculo, e seria melhor concluir que estes três requisitos são igualmente importantes, o que poderia ter sido o seu instinto desde o início.
Certifique-se sempre de que o esforço gasto na priorização seja justificado pelo próprio valor de uma correta priorização. Portanto, mais uma vez, mantenha os objetivos em mente e lembre-se do Princípio 1: orientação a valor.

6.9 Leitura adicional

Os livros didáticos de Pohl [Pohl2010], Davis [Davi2005], Hull, Jackson e Dick [HuJD2011], van Lamsweerde [vLam2009] e Wiegers e Beatty [WiBe2013] fornecem uma visão abrangente do gerenciamento de requisitos. Outras informações sobre o tópico de gerenciamento de requisitos estão consolidadas no CPRE Advanced Level Handbook for Requirements Management de Bühne e Herrmann [BuHe2019].

Cleland-Huang, Gotel e Zisman [ClGZ2012] oferecem um tratamento profundo de rastreabilidade.

Olson [Olso2014] e Wiegers [Wieg1999] lidam com técnicas de priorização.

7 Ferramentas de Suporte

Um Engenheiro de Requisitos precisa de ferramentas para a prática de suas habilidades – assim como um carpinteiro precisa de suas ferramentas, lápis, um martelo, uma serra e uma furadeira para projetar e realizar uma peça de mobiliário. Sem ferramentas, é difícil ou impossível registrar os requisitos, trabalhar em conjunto sobre os requisitos e estar no controle dos requisitos.

Este capítulo examina os diferentes tipos de ferramentas de Engenharia de Requisitos (ER) disponíveis e os aspectos que precisam ser levados em consideração para introduzir as ferramentas de Engenharia de Requisitos em uma organização.

7.1 Ferramentas na Engenharia de Requisitos

A Engenharia de requisitos é uma tarefa difícil sem o apoio de ferramentas. São necessárias ferramentas para apoiar as tarefas e atividades de Engenharia de Requisitos. As ferramentas existentes concentram-se no apoio a tarefas específicas, tais como documentação de requisitos ou controle do processo de ER, e raramente em todas as tarefas e atividades do processo de Engenharia de Requisitos. Portanto, não é surpreendente que o Engenheiro de Requisitos tenha um conjunto de ferramentas à sua disposição para apoiar os vários componentes no processo de engenharia de requisitos – assim como o carpinteiro precisa de várias ferramentas (por exemplo, projeto auxiliado por computador (CAD)) para projetar um móvel e precisa de ferramentas como uma serra, raspador e lixa para realizá-lo.

As ferramentas são apenas um auxílio ao processo de Engenharia de Requisitos e ao Engenheiro de Requisitos, e tais ferramentas são chamadas de ferramentas CASE (computer-aided software engineering). As ferramentas CASE apoiam uma tarefa específica no processo de produção de software [Fugg1993].

Diferenciamos diversos tipos de ferramentas que suportam os seguintes aspectos da Engenharia de Requisitos:

- Gerenciamento de requisitos

  As ferramentas nesta categoria têm as propriedades necessárias para apoiar as atividades e tópicos descritos no Capítulo 6. Com este tipo de ferramentas, é possível estabelecer mais controle sobre o processo de Engenharia de Requisitos. Os requisitos estão sujeitos a mudanças e em um ambiente onde isso acontece com frequência, uma ferramenta com as propriedades relevantes é indispensável. Ferramentas de apoio nesta categoria:

  - Definição e armazenamento de atributos de requisitos para identificar e coletar dados sobre produtos de trabalho e requisitos, conforme descrito na Seção 6.5
  - Facilitação e documentação da priorização dos requisitos (Seção 6.8)
  - Gerenciamento do ciclo de vida, controle de versões, configurações e linhas de base, conforme descrito nas Seções 6.2, 6.3, e 6.4

    - Localização e rastreamento de requisitos, bem como defeitos nos requisitos e produtos de trabalho (Seção 6.6)
    - Gerenciamento de mudanças de requisitos; como aprendemos na Seção 6.7, as mudanças são inevitáveis e têm de ser cuidadosamente gerenciadas
- Processo de Engenharia de Requisitos

  Para suportar e controlar o processo de Engenharia de Requisitos, são necessárias informações para permitir que o processo seja ajustado ou melhorado. Este tipo de ferramenta pode:

    - Medir e informar sobre o processo e o fluxo de trabalho da Engenharia de Requisitos
    - Estas informações ajudam a melhorar o processo de Engenharia de Requisitos e reduzem desperdícios.
    - Medir e informar sobre a qualidade do produto
    - Estas informações ajudam a encontrar defeitos e falhas, que por sua vez podem ser usadas para melhorar a qualidade do produto.
- Documentação de conhecimento sobre os requisitos

  A quantidade de conhecimento (e requisitos) acumulada em um projeto pode ser enorme. Além disso, uma grande quantidade de conhecimento é acumulada sobre um produto durante seu ciclo de vida. Todas as informações relevantes devem ser cuidadosamente documentadas para permitir o seguinte:

    - Compartilhamento e criação de um entendimento comum dos requisitos
    - Assegurar os requisitos como uma obrigação legal
    - Uma visão geral e conhecimento sobre os requisitos
- Modelagem de requisitos

  Como aprendemos na Seção 3.4.1.6, a representação de requisitos, tanto como diagramas quanto em linguagem natural, agrega os pontos fortes de ambas as formas de notação. Uma ferramenta que pode modelar requisitos permite a você:

    - Estruturar seus próprios pensamentos; ele pode ser usado como uma ajuda para pensar
    - Especificar os requisitos em uma linguagem mais formal do que os requisitos textuais, com todos os benefícios que isto traz
- Colaboração em Engenharia de Requisitos

  Quando várias pessoas e disciplinas trabalham no mesmo projeto, uma ferramenta pode apoiar e possibilitar essa colaboração, especialmente no mundo em que vivemos atualmente, onde cada vez mais atividades são realizadas localmente (em casa). Este tipo de ferramenta apoia a elicitação, documentação e gerenciamento de requisitos.

- Teste e/ou Simulação de Requisitos

  As ferramentas estão se tornando cada vez mais sofisticadas. Cada vez mais opções estão sendo desenvolvidas para testes e/ou simulação de requisitos com

antecedência. Isto permite melhor prever se os requisitos propostos terão o efeito pretendido.

As ferramentas disponíveis são muitas vezes uma mistura das anteriores. Como mencionado anteriormente, diferentes ferramentas podem precisar ser combinadas para apoiar adequadamente a Engenharia de Requisitos. Se forem utilizadas diferentes ferramentas, é importante prestar atenção à integração entre elas e à interação com outras aplicações e sistemas, a fim de garantir uma operação sem solução de continuidade.

Algumas vezes, outros tipos de ferramentas (por exemplo pacotes de aplicativos para escritório e serviços ou ferramentas de rastreamento de problemas) são usados para documentar ou gerenciar requisitos. Entretanto, essas ferramentas têm suas limitações e devem ser usadas somente quando os Engenheiros de Requisitos e os stakeholders estiverem no controle do processo de ER e os requisitos estiverem alinhados. Caso contrário, este é um grande risco no processo de ER, pois tais ferramentas não suportam nenhuma atividade de gerenciamento de requisitos.

7.2 Introduzindo Ferramentas

Selecionar uma ferramenta de ER não é diferente de selecionar uma ferramenta para qualquer outro propósito. Você deve descrever os objetivos, contexto e requisitos antes de selecionar e implementar a(s) ferramenta(s) apropriada(s).

As ferramentas são apenas um auxílio ao processo de Engenharia de Requisitos e ao Engenheiro de Requisitos. Eles não resolvem questões organizacionais ou humanas. Imagine que, junto com seus colegas, você queira documentar os requisitos de maneira uniforme. As ferramentas podem suportar isto, por exemplo, com um modelo em uma ferramenta de processamento de texto ou página wiki. Isto não garante que todos os seus colegas adotem este método de trabalho, nem assegura que seus colegas tenham a disciplina para registrar e gerenciar seus requisitos desta forma. O que pode ajudar é fazer acordos uns com os outros, verificar se os acordos estão sendo cumpridos, e ser capaz de se comunicar uns com os outros se os acordos não forem cumpridos. Uma ferramenta não vai ajudá-lo com isto. A introdução de uma ferramenta de Engenharia de Requisitos requer responsabilidades e procedimentos claros de Engenharia de Requisitos.

Uma ferramenta pode ajudá-lo a configurar seu processo de Engenharia de Requisitos de forma eficaz e eficiente. As ferramentas frequentemente fornecem uma estrutura baseada na experiência das melhores práticas. do mercado. Essas estruturas podem então ser customizadas para se adequarem à situação.

Como aprendemos nos capítulos anteriores, as atividades centrais da Engenharia de Requisitos não são processos autônomos.

A seleção das ferramentas de ER apropriadas começa com a definição dos objetivos e/ou problemas que você deseja resolver no processo de ER. O próximo passo é determinar o contexto do sistema (neste caso, o conjunto de ferramentas). Considere os aspectos do contexto – ou seja, stakeholders, processos, eventos etc., e aplique suas habilidades de

Engenharia de Requisitos para especificar os requisitos para as ferramentas de ER. Pratique o que você prega.

As próximas seções descrevem alguns dos aspectos que devem ser levados em consideração ao introduzir uma (nova) ferramenta de Engenharia de Requisitos em sua organização.

7.2.1 Considerar todos os custos do ciclo de vida além dos custos de licença

Os custos mais óbvios, tais como custos de compra ou de licenciamento, são geralmente levados em consideração. Além disso, custos menos visíveis também devem ser levados em conta, como o uso de recursos na implementação, operação e manutenção da ferramenta.

7.2.2 Considerar os Recursos Necessários

A especificação dos requisitos e a supervisão do processo de seleção requerem recursos adicionais, além dos custos mencionados na seção anterior. As pessoas necessárias para orientar o processo de seleção, Engenheiros de Requisitos, recursos de hardware e outros recursos não devem ser negligenciados. Depois que a ferramenta for colocada em uso, também poderão ser necessários recursos para manutenção e suporte ao usuário.

7.2.3 Evitar riscos através da execução de projetos-piloto

A introdução de uma nova ferramenta pode ameaçar o controle sobre a base de requisitos atuais. Um caos de requisitos pode surgir porque há uma transição do velho método de trabalho e/ou ferramentas para o novo método de trabalho e/ou ferramentas. A introdução de uma nova ferramenta durante um projeto existente levará irrevogavelmente a um atraso na entrega dos requisitos e até mesmo do projeto.

A introdução de uma nova ferramenta, possivelmente com um método de trabalho diferente, deve ser testada em pequena escala, onde os riscos e o impacto permanecem gerenciáveis. Há várias maneiras de se fazer isso:

- Aplicar a ferramenta a um projeto/sistema não-crítico
- Utilizar a ferramenta de forma redundante ao lado de um projeto existente
- Aplicar a ferramenta a uma situação/projeto fictício
- Importar/copiar os requisitos de um projeto que já tenha sido concluído

Quando se chega ao ponto em que a ferramenta atende às metas e requisitos estabelecidos, ela pode ser implementada mais amplamente dentro da organização ou em outros projetos.

7.2.4 Avaliar a ferramenta de acordo com critérios definidos

Selecionar a ferramenta apropriada pode ser uma tarefa difícil. A verificação extensiva do cumprimento dos objetivos e requisitos é uma abordagem padrão na Engenharia de

Requisitos. Uma abordagem sistemática que avalia a ferramenta a partir de diferentes perspectivas também contribui para fazer a escolha certa. As seguintes perspectivas podem ser consideradas:

- Perspectiva do projeto

  Este ponto de vista destaca os aspectos de gerenciamento do projeto. A ferramenta apoia o projeto e as informações necessárias no projeto?

- Perspectiva do processo

  Esta perspectiva verifica o suporte ao processo de Engenharia de Requisitos. A ferramenta suporta suficientemente o processo de ER? Pode ser suficientemente adaptado ao processo de ER existente e ao método de trabalho?

- Perspectiva do usuário

  Esta perspectiva verifica o grau de aplicação por parte dos usuários da ferramenta. Esta é uma visão importante porque se os usuários não ficarem satisfeitos com a ferramenta, aumenta o risco de que a ferramenta não seja aceita. A ferramenta apoia suficientemente a gestão de autorização de usuários e grupos? É suficientemente fácil e intuitivo de usar?

- Perspectiva do produto

  As funcionalidades oferecidas pela ferramenta são verificadas a partir deste ângulo. Os requisitos são adequadamente cobertos pela ferramenta? Onde os dados são armazenados? Existe um roteiro com as extensões funcionais para a ferramenta? A ferramenta ainda é suportada pelo fornecedor?

- Perspectiva do fornecedor

  Com esta perspectiva, o foco está no serviço e na confiabilidade do fornecedor. Onde está localizado o fornecedor? Como é organizado o suporte para esta ferramenta?

- Perspectiva econômica

  Esta perspectiva analisa o caso de negócio: a ferramenta proporciona benefícios suficientes em relação aos seus custos? Quais são os custos (de administração) para a compra e manutenção? O que a ferramenta fornece para o processo de ER? É necessário um contrato de manutenção (separado)?

- Perspectiva de arquitetura

  Esta perspectiva avalia como a ferramenta se encaixa na organização (TI). A tecnologia aplicada é adequada à organização? A ferramenta pode ser suficientemente interligada a outros sistemas? A ferramenta se encaixa no cenário de TI e está de acordo com as restrições arquitetônicas?

7.2.5 Instruir os funcionários sobre o uso da ferramenta

Uma vez selecionada uma ferramenta, os usuários devem se familiarizar com as oportunidades que a ferramenta pode agregar ao processo de Engenharia de Requisitos. Os usuários – isto é, os Engenheiros de Requisitos – devem ser treinados em como utilizar a ferramenta no processo de Engenharia de Requisitos existente. Se os usuários não estiverem suficientemente treinados, isto pode significar que nem todos os benefícios da ferramenta são utilizados. Na verdade, é possível que a ferramenta seja utilizada incorretamente, com todas as consequências associadas.

O processo de Engenharia de Requisitos também pode ser alterado devido à ferramenta selecionada. Aspectos no processo de Engenharia de Requisitos que antes não eram possíveis com uma nova ferramenta: por exemplo, gerenciamento adequado de versões, modelagem de requisitos etc. Isto pode significar que novos procedimentos são acordados, templates são adaptados ou aplicados, mudanças são feitas no método de trabalho, e assim por diante. O envolvimento do Engenheiro de Requisitos nesta mudança contribui para o sucesso do aceite da ferramenta.

7.3 Leitura adicional

A literatura a seguir pode ser consultada para uma visão geral das ferramentas disponíveis e avaliações de ferramentas. Juan M. Carrillo de Gea et. al. fornecem uma visão abrangente do papel das ferramentas de engenharia de requisitos [dGeA2011]. O artigo de Barbara Kitchenham, Stephen Linkman, David Law [KiLL1997] descreve e valida um método para avaliação sistemática de ferramentas. Se você estiver procurando uma ferramenta de ER, uma lista abrangente de ferramentas para Engenharia de Requisitos é fornecida no site da Volere [Vole2020] ou em [BiHe2020].

8 Referências

[Alex2005] Ian F. Alexander: A Taxonomy of Stakeholders – Human Roles in System Development. International Journal of Technology and Human Interaction 2005, 1(1), 23–59.

[AnPC1994] Annie I. Antón, W. Michael McCracken, Colin Potts: Goal Decomposition and Scenario Analysis in Business Process Reengineering. CAiSE (Conference on Advanced Information Systems Engineering), 1994, 94–104.

[Armo2004] Philip G. Armour. The Laws of Software Process: A New Model for the Production and Management of Software. Boca Raton, Fl.: CRC Press, 2004.

[Axelos2019] Axelos: ITIL Foundation: ITIL 4 Edition. Axelos Ltd., 2019.

[BaBo2014] Stéphane Badreau, Jean-Louis Boulanger: Ingénierie des Exigences. Paris: Dunod, 2014 (in French).

[BeeA2001] Kent Beck et al.: Principles behind the Agile Manifesto. http://agilemanifesto.org/principles.html, 2001. Última visita Agosto 2022.

[BiHe2020] Andreas Birk, Gerald Heller: List of Requirements Management Tools. https://makingofsoftware.com/resources/list-of-rm-tools/, 2020, Última visita Agosto 2020.

[Boeh1981] Barry W. Boehm: Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[BoRJ2005] Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide, 2nd edition. Reading, MA: Addison-Wesley, 2005.

[Bour2009] Lynda Bourne: Stakeholder Relationship Management – A Maturity Model for Organisational Implementation. Farnham: Gower, 2009.

[BuHe2019] Stan Bühne, Andrea Herrmann: Handbook Requirements Management according to the IREB Standard – Education and Training for the IREB Certified Professional for Requirements Engineering Qualification Advanced Level Requirements Management. Karlsruhe: IREB. https://www.ireb.org/downloads/#cpre-advanced-level-requirements-management-handbook, 2019. Última visita Agosto 2022.

[CaDJ2014] Dante Carrizo, Oscar Dieste, Natalia Juristo: Systematizing Requirements Elicitation Technique Selection. Information and Software Technology 2014, 56(6), 644–669.

[Chen1976] Peter P.-S. Chen: The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems 1976, 1(1), 9–36.

[ClGZ2012] Jane Cleland-Huang, Olly Gotel, Andrea Zisman (eds.): Software and Systems Traceability. London: Springer, 2012.

[Cock2001] Alistair Cockburn: Writing Effective Use Cases. Boston: Addison-Wesley, 2001.

[Cohn2004] Mike Cohn: User Stories Applied: For Agile Software Development. Boston: Addison-Wesley, 2004.

[Cohn2010] Mike Cohn: Succeeding with Agile: Software Development Using Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010.

[CoWe1998] Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management. ACM Computing Surveys 1998, 30(2), 232–282.

[DaTW2012] Marian Daun, Bastian Tenbergen, Thorsten Weyer: Requirements Viewpoint. In: K. Pohl, H. Hönninger, R. Achatz, M. Broy: Model-Based Engineering of Embedded Systems, Heidelberg: Springer, 2012.

[Davi1993] Alan M. Davis: Software Requirements – Objects, Functions, and States. 2nd Edition, Englewood Cliffs, New Jersey: Prentice Hall, 1993.

[Davi1995] Alan M. Davis: 201 Principles of Software Development. New York: McGraw-Hill, 1995.

[Davi2005] Alan M. Davis: Just Enough Requirements Management – Where Software Development Meets Marketing. New York: Dorset House, 2005.

[DeBo2005] Edward De Bono: De Bono's Thinking Course (Revised Edition), Barnes & Noble Books, 2005.

[DeCo2007] Design Council: 11 Lessons: A Study of the Design Process. https://www.designcouncil.org.uk/resources/report/11-lessons-managing-design-global-brands, 2007. Última visita Agosto 2022.

[dGeA2011] Juan M. Carrillo de Gea, Joaquín Nicolás, José L. Fernandez-Alemán, Ambrosio Toval, Christof Ebert, Aurora Vizcaíno: Requirements Engineering Tools. IEEE Software 2011, 28(4), 86–91.

[DeMa1978] Tom DeMarco: Structured Analysis and System Specification. New York: Yourdon Press, 1978.

[DIN66001] DIN 66001:1983-12: Information processing; graphical symbols and their application. Deutsches Institut für Normung e.V., Berlin, 1983 (in German).

[Eber2014] Christof Ebert: Systematisches Requirements Engineering, 5. Auflage. Heidelberg: dpunkt 2014 (in German).

[Fowl1996] Martin Fowler: Analysis Patterns: Reusable Object Models. Reading, MA: Addison-Wesley, 1996.

[FLCC2016] Xavier Franch, Lidia Lopez, Carlos Cares, Daniel Colomer. (2016). A estrutura i* para modelagem orientada a metas. Em Domain Specific Conceptual Modeling, Springer, 485-506.

[Fugg1993] Alfonso Fuggetta: A Classification of CASE Technology. IEEE Computer 1993, 26(12), 25–38.

[GaWe1989] Donald C. Gause and Gerald M. Weinberg: Exploring Requirements: Quality before Design. New York: Dorset House, 1989.

[GFPK2010] Tony Gorschek, Samuel Fricker, Kenneth Palm, and Steven A. Kunsman: A Lightweight Innovation Process for Software-Intensive Product Development. IEEE Software 2010, 27(1), 37–45.

[GGJZ2000] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, Pamela Zave: A Reference Model for Requirements and Specifications. IEEE Software 2000, 17(3), 37–43.

[GHJV1994] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Pattern – Elements of Reusable Object-Oriented Software. Reading, Mass.: Addison-Wesley, 1994.

[Gilb1988] Tom Gilb: Principles of Software Engineering Management. Reading, Mass.: Addison Wesley, 1988.

[Glas1999] Friedrich Glasl: Confronting Conflict – A First-Aid Kit for Handling Conflict. Stroud, Gloucestershire: Hawthorn Press, 1999.

[GlFr2015] Martin Glinz and Samuel A. Fricker: On Shared Understanding in Software Engineering: An Essay. Computer Science – Research and Development 2015, 30(3–4), 363–376.

[Glin2007] Martin Glinz: On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference, Delhi, India, 2007, 21–26.

[Glin2008] Martin Glinz: A Risk-Based, Value-Oriented Approach to Quality Requirements. IEEE Software 2008, 25(2), 34–41.

[Glin2016] Martin Glinz: How Much Requirements Engineering Do We Need? Softwaretechnik-Trends 2016, 36(3), 19–21.

[Glin2019] Martin Glinz: Introdução à Engenharia de Requisitos. Course Notes, University of Zurich, 2019. https://www.ifi.uzh.ch/en/rerg/courses/archives/hs19/re-i.html#resources. Última visita Agosto 2022.

[Glin2020] Martin Glinz: A Glossary of Requirements Engineering Terminology. Versão 2.0. https://www.ireb.org/downloads/#cpre-glossary, 2020. Última visita Agosto 2022.

[GlWi2007] Martin Glinz and Roel Wieringa: Stakeholders in Requirements Engineering (Guest Editors' Introduction). IEEE Software 2007, 24(2), 18–20.

[GoFi1994] Orlena Gotel, Anthony Finkelstein: An Analysis of the Requirements Traceability Problem. 1st International Conference on Requirements Engineering, Colorado Springs, 1994, 94–101.

[GoRu2003] Rolf Goetz, Chris Rupp: Psychotherapy for System Requirements. 2nd IEEE International Conference on Cognitive Informatics (ICCI'03), London, 2003, 75–80.

[Gott2002] Ellen Gottesdiener: Requirements by Collaboration: Workshops for Defining Needs, Boston: Addison-Wesley Professional, 2002.

[GreA2017] Eduard C. Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitzá Guzmán, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, Melanie Stade: The Crowd in Requirements Engineering – The Landscape and Challenges. IEEE Software 2017, 34(2), 44–52.

[Greg2016] Sarah Gregory: "It Depends": Heuristics for "Common Enough" Requirements. Keynote speech at REFSQ 2016, Essen, Germany, 2016.

[GRL2020] Goal oriented Requirement Language. University of Toronto, Canada https://www.cs.toronto.edu/km/GRL. Última visita Agosto 2022.

[GrSe2005] Paul Grünbacher, Norbert Seyff: Requirements Negotiation. In A. Aurum, C. Wohlin (eds.): Engineering and Managing Software Requirements. Berlin: Springer, 2005, 143–162.

[Hare1988] David Harel. On Visual Formalisms. Communications of the ACM 1988, 31(5), 514–530.

[HoSch2020] Stefan Hofer, Henning Schwentner: Domain Storytelling — A Collaborative Modeling Method. Available from Leanpub, http://leanpub.com/domainstorytelling. Última visita Agosto 2022.

[HuJD2011] Elizabeth Hull, Ken Jackson, Jeremy Dick: Requirements Engineering. 3rd ed., Berlin: Springer: 2011.

[Hump2017] Aaron Humphrey: User Personas and Social Media Profiles. Persona Studies 2017, 3(2), 13–20.

[IEEE830] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830–1998, 1998.

[ISO19650] ISO 19650. Organization and Digitization of Information about Buildings and Civil Engineering Works, including Building Information Modelling (BIM)– Information Management Using Building Information Modelling – Part 1 and 2, 2018.

[ISO5807] ISO/IEC/IEEE 1985–02: Information processing; Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. International Organization for Standardization, Geneva, 1985.

[ISO25010] ISO/IEC/IEEE 25010:2011: Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Organization for Standardization, Geneva, 2011.

[ISO29148] ISO/IEC/IEEE 29148: Systems and Software Engineering – Life Cycle Processes – Requirements Engineering. International Organization for Standardization, Geneva, 2018.

[Jack1995] Michael Jackson: Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices. New York: ACM Press, 1995.

[Jack1995b] Michael Jackson: The World and the Machine. 17th International Conference on Software Engineering 1995 (ICSE 1995), 287–292.

[Jaco1992] Ivar Jacobson: Object-oriented software engineering: a use case driven approach. New York: ACM Press, 1992.

[JaSB2011] Ivar Jacobson, Ian Spence, Kurt Bittner: Use Case 2.0: The Guide to Succeeding with Use Cases (Guia para obter sucesso com casos de uso). Ivar Jacobson International SA, 2011.

[KiLL1997] Barbara Kitchenham, Stephen Linkman, David Law: DESMET: A Methodology for Evaluating Software Engineering Methods and Tools. Computing & Control Engineering Journal 1997, 8(3), 120–126.

[KSTT1984] Noriaki Kano, Nobuhiku Seraku, Fumio Takahashi, Shinichi Tsuji: Attractive Quality and Must-Be Quality. Hinshitsu (Quality – Journal of the Japanese Society for Quality Control) 1984, 14(2), 39–48 (in Japanese).

[Laue2002] Søren Lauesen: Software Requirements: Styles and Techniques. London: Addison-Wesley, 2002.

[LaWE2001] Brian Lawrence, Karl Wiegers, and Christof Ebert: The Top Risks of Requirements Engineering. IEEE Software 2001, 18(6), 62–63.

[LiOg2011] Jeanne Liedtka, Tim Ogilvie: Designing for Growth: A Design Thinking Tool Kit for Managers. New York: Columbia University Press, 2011.

[LiSS1994] Odd I. Lindland, Guttorm Sindre, Arne Sølverg: Understanding Quality in Conceptual Modeling. IEEE Software 1994, 11(2), 42–49.

[LiSZ1994] Horst Lichter, Matthias Schneider-Hufschmidt, Heinz Züllighoven: Prototyping in Industrial Software Projects – Bridging the Gap Between Theory and Practice. IEEE Transactions on Software Engineering 1994, 20(11), 825–832.

[LiQF2010] Soo Ling Lim, Daniele Quercia, Anthony Finkelstein: StakeNet: Using Social Networks to Analyse the Stakeholders of Large-Scale Software Projects. 32nd International Conference on Software Engineering (ICSE 2010), 2010, 295–304.

[MaGR2004] Neil Maiden, Alexis Gizikis, Suzanne Robertson: Provoking Creativity: Imagine What Your Requirements Could Be Like. IEEE Software 2004, 21(5), 68–75.

[Math2019] Joseph Mathenge: Change Control Board vs Change Advisory Board: What's the Difference? https://www.bmc.com/blogs/change-control-board-vs-change-advisory-board, Nov. 22, 2019. Última visita Agosto 2022.

[McIn2016] John McIntyre: MoSCoW or Kano Models – How Do You Prioritize? https://www.hotpmo.com/management-models/moscow-kano-prioritize, Oct. 20, 2016. Última visita Agosto 2022.

[MNJR2016] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe: Toward Data-Driven Requirements Engineering. IEEE Software 2016, 33(1), 48–54.

[Moor2014] Christopher W. Moore: The Mediation Process – Practical Strategies for Resolving Conflicts, 4th edition. Hoboken, NJ: John Wiley & Sons, 2014.

[MWHN2009] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak: Easy Approach to Requirements Syntax (EARS). 17th IEEE International Requirements Engineering Conference (RE'09), Atlanta, Georgia, 2009, 317–322.

[NuKF2003] Bashar Nuseibeh, Jeff Kramer, Anthony Finkelstein: ViewPoints: Meaningful Relationships are Difficult! 25th International Conference on Software Engineering (ICSE'03), Portland, Oregon, 2003, 676–681.

[OleA2018] K. Olsen et al.: Certified Tester, Foundation Level Syllabus – Version 2018. International Software Testing Qualifications Board, 2018.

[Olso2014] David Olson: Matrix Prioritization. http://www.bawiki.com/wiki/Matrix-Prioritization.html, 2014. Última visita Agosto 2022.

[OMG2013] Object Management Group: Business Process Model and Notation (BPMN), version 2.0.2. OMG document, formal/2013-12-09. https://www.omg.org/spec/BPMN/. Última visita Agosto 2022.

[OMG2017] Object Management Group: OMG Unified Modeling Language (OMG UML), version 2.5.1. OMG document, formal/2013-12-09. https://www.omg.org/spec/UML/About-UML/. Última visita Agosto 2022.

[OMG2018] Object Management Group: OMG Systems Modeling Language (OMG SysML™), version 1.6. OMG document ptc/2018-12-08. https://www.omg.org/spec/SysML/About-SysML/. Última visita Agosto 2022.

[Osbo1948] Alex F. Osborn: Your Creative Power: How to Use Imagination. C. Scribner's Sons, 1948. (Accessed as digital reprint: Read Books Ltd. (epub eBook), April 2013).

[Pich2010] Roman Pichler: Agile Product Management with Scrum – Creating Products that Customers Love, Boston: Addison-Wesley, 2010.

[Pohl2010] Klaus Pohl: Requirements Engineering: Fundamentals, Principles, and Techniques (Fundamentos, Princípios e Técnicas). Berlin-Heidelberg: Springer, 2010.

[PoRu2015] Klaus Pohl, Chris Rupp: Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam, (2nd ed). Rocky Nook, Santa Barbara, 2015.

[Rein1997] Donald G. Reinertsen: Managing the Design Factory – A Product Developer's Toolkit. The Free Press, 1997.

[Rein2009] Donald G. Reinertsen: The Principles of Product Development Flow: Second Generation Lean Product Development. Redondo Beach, Ca.: Celeritas Publishing, 2009.

[Ries2011] Eric Ries: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. New York: Crown Business, 2011.

[Robe2001] S. Ian Robertson: Problem Solving. Hove, East Sussex: Psychology Press, 2001.

[RoRo2012] Suzanne Robertson and James Robertson: Mastering the Requirements Process: Getting Requirements Right. 3rd edition. Boston: Addison-Wesley, 2012.

[RuJB2004] James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual, 2nd edition. Reading, MA: Addison Wesley, 2004.

[Rupp2014] Chris Rupp: Requirements-Engineering und Management, 6. Auflage. München: Hanser, 2014 (in German).

[SoSa1998] Ian Sommerville and Pete Sawyer: Requirements Engineering: A Good Practice Guide. Chichester: John Wiley & Sons, 1997.

[SwBa1982] William Swartout and Robert Balzer: On the Inevitable Intertwining of Specification and Implementation. Communications of the ACM 1982, 25(7), 438–440.

[Verd2014] Dave Verduyn: Discovering the Kano Model, in: Kano model, https://www.kanomodel.com/discovering-the-kano-model, 2014. Última visita Agosto 2022.

[vLam2009] Axel van Lamsweerde: Requirements Engineering: From System Goals to UML Models to Software Specifications. Chichester: John Wiley & Sons, 2009.

[Vole2020] Volere Requirements Resources: https://www.volere.org. Última visita Agosto 2022.

[WiBe2013] Karl Wiegers and Joy Beatty: Software Requirements. 3rd edition. Redmond, Wa.: Microsoft Press, 2013.

[Wieg1999] Karl E. Wiegers: First Things First: Prioritizing Requirements. https://www.processimpact.com/articles/prioritizing.pdf, 1999. Última visita Agosto 2022.

[ZoCo2005] Didar Zowghi, Chad Coulin: Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In A. Aurum, C. Wohlin (eds.) Engineering and Managing Software Requirements. Berlin: Springer, 2005, 19–46.