

Unidade II

3 ENGENHARIA DE REQUISITOS

3.1 Introdução à Engenharia de Requisitos

Como vimos, engenharia é um conjunto de métodos, procedimentos e ferramentas que tem por objetivo resolver um determinado problema.

Requisitos, segundo Sommerville (2010), são serviços que um sistema deve prestar e suas restrições de funcionamento. Eles devem necessariamente refletir as necessidades do cliente.

Engenharia de Requisitos é um conjunto de métodos, procedimentos e ferramentas que tem por objetivo descobrir, analisar, documentar, verificar e validar esses requisitos (SOMMERVILLE, 2010).

Se olharmos sob o ângulo da Engenharia de Software, podemos dizer que a Engenharia de Requisitos é um ramo da Engenharia de Software que envolve, dentro do ciclo de vida de um *software*, atividades relacionadas a requisitos (KOTONYA; SOMMERVILLE, 1998).

Entre os desafios da Engenharia de Requisitos, Kotonya e Sommerville (1998) destacam:

- A definição de requisitos inconsistentes leva a problemas que se estendem por todo ciclo de vida do *software*.
- A Engenharia de Requisitos proporciona estimar com maior precisão o tempo e o custo de um projeto. Requisitos mal definidos geram um projeto com maior custo e maior tempo de desenvolvimento.
- A Engenharia de Requisitos proporciona melhor gerência das mudanças, comuns em um projeto. Em contrapartida, falhas nesse processo geram sistemas de *software* com altos custos de manutenção.

Um sistema de *software* que tenha todos os problemas citados decorrerá, invariavelmente, em um cliente insatisfeito, ou seja, um projeto fracassado.

Para termos total noção da importância da Engenharia de Requisitos, vejamos um estudo proposto pelo grupo de pesquisas The Standish Group (1995).

As estatísticas mostram, e nos motivam a ver, como a Engenharia de Requisitos pode ser importante dentro do ciclo de vida de um projeto. O estudo analisou um total de aproximadamente 8.400 projetos com o objetivo de mensurar as causas de sucesso e fracasso.

Observação

O The Standish Group vem coletando dados de projetos nos Estados Unidos desde 1994 e mapeando, entre outras coisas, as causas de sucesso e de fracasso.

A figura a seguir mostra, em números aproximados, a situação final dos projetos analisados.

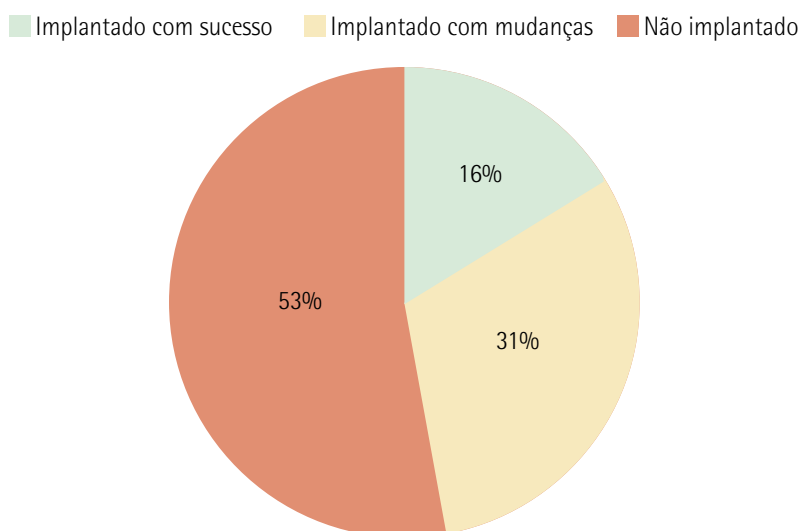


Figura 10 – Percentual de Projetos Finalizados

Apenas 16% dos projetos que foram iniciados terminaram sendo implantados com sucesso. Os dados mostram ainda que 31% dos projetos iniciados tiveram sua implantação concluída, mas com ressalvas: o projeto formatado originalmente não foi o mesmo entregue no final. Entre os problemas encontrados, estão:

- projetos foram entregues fora do prazo acordado inicialmente;
- projetos foram entregues fora do custo orçado inicialmente;
- projetos foram entregues com uma quantidade de funcionalidades menor do que a prevista inicialmente.

E, por fim, 53% dos projetos não foram entregues, o que quer dizer que foram cancelados em algum ponto do processo. O quadro a seguir mostra os principais fatores de atraso e de não atendimento às expectativas.

Quadro 3 – Principais fatores de insucesso

Fator	Percentual
Baixo envolvimento do cliente/usuário	13%
Requisitos e especificações incompletas	12%
Mudanças nos requisitos	12%
Falta de apoio executivo	7%
Uso de tecnologia inadequada	7%
Recursos e pessoal inadequado	6%
Cronograma fora da realidade	6%

Adaptado de: The Standish Group (1995, p. 9).

Note que fatores que envolvem diretamente a Engenharia de Requisitos – baixo envolvimento do cliente/usuário; requisitos e especificações incompletas; mudanças nos requisitos – somam ao todo 37% dos fatores de insucesso.

O quadro a seguir mostra os principais fatores de sucesso dos projetos de *software*. Observe a importância da Engenharia de Requisitos nos fatores de sucesso, nos quais temos participação direta ou indireta em todos os fatores.

Quadro 4 – Principais fatores de sucesso

Fator	Percentual
Envolvimento do cliente/usuário	16%
Apoio executivo e gerencial	14%
Enunciado claro dos requisitos	12%
Planejamento apropriado	9,6%
Expectativas realistas	8,2%

Adaptado de: The Standish Group (1995, p. 8).

Como vimos, a Engenharia de Requisitos é um importante fator de sucesso para um projeto, mas quem são as pessoas interessadas, dentro do projeto, pelos requisitos?

Em primeiro lugar, o mais interessado nos requisitos é o cliente e o usuário, mas eles possuem papéis diferentes. O cliente é o contratante do projeto, o usuário é quem vai, efetivamente, usar o sistema, quem detém o maior conhecimento do negócio. É importante que o cliente e o usuário sejam parte integrante do projeto, que se envolvam e se motivem com o projeto.

Além do usuário e do cliente, são considerados interessados pelos requisitos: analista de sistemas, desenvolvedores, arquitetos e gerentes de projeto.

Cada papel envolvido no projeto pode ser considerado um interessado nos requisitos e na Engenharia de Requisitos, o que muda é apenas o ponto de vista de cada um.

O desenvolvedor, por exemplo, tem uma visão técnica de implementação de um requisito, enquanto o arquiteto tem uma visão abstrata sobre como os problemas serão resolvidos da melhor forma dentro da solução (SOMMERVILLE, 2010).



Observação

Stakeholder é o nome dado pelos diversos interessados ou envolvidos no projeto. Se olharmos pelo ângulo da Engenharia de Requisitos, *stakeholder* são todos os interessados ou envolvidos pelos requisitos.

3.2 Classificação de requisitos

Diante da necessidade de expressar requisitos para diversas pessoas, com diversos pontos de vista, Sommerville (2010) sugere a separação dos requisitos a partir do seu nível de descrição. O autor sugere a classificação em dois níveis:

- Requisitos de usuário: segundo Sommerville (2010, p. 83), requisitos de usuário são "declarações em linguagem natural com diagramas de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais ele deve operar". Eles devem ser direcionados aos clientes, usuários, gerentes de projeto e arquitetos de sistema, pois definem em alto nível as necessidades, o que define, entre outras coisas, o escopo do projeto.
- Requisitos de sistema: segundo Sommerville (2010), requisitos de sistema são "descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de *software*". Eles devem ser direcionados a usuários, arquitetos de sistema e desenvolvedores, pois podem definir uma sequência de implementação, o que influencia diretamente na solução dada.

Voltando ao nosso exemplo do cliente de autoatendimento bancário, vejamos o quadro a seguir, no qual está representada a descrição de um requisito de usuário e seus requisitos de sistema.

Quadro 5 – Requisitos de usuário x sistema

Requisito de usuário	Requisito de sistema
O sistema de autoatendimento deve permitir ao cliente efetuar um saque de conta-corrente em dinheiro.	<p>1.1 O acesso ao sistema de autoatendimento será feito mediante leitura do cartão com <i>chip</i> e digitação da senha de segurança</p> <p>1.2 O sistema deverá exibir a opção de saldo de conta-corrente, se o cliente possuir saldo e/ou limite para efetuar a operação</p> <p>1.3 Se o cliente não possuir saldo e/ou limite para efetuar a operação, ele deverá ser informado da situação.</p> <p>1.4 O cliente somente poderá efetuar saque de quantias que sejam múltiplas das cédulas existentes no equipamento.</p>

1.5 As cédulas deverão ser dispensadas e o cliente deverá ser orientado a retirá-las no local adequado.

1.6 Caso o cliente não retire as cédulas em um tempo mínimo de 30 segundos, elas deverão ser automaticamente recolocadas no terminal.

Note que a diferença no nível de descrição está ligada à quantidade e à qualidade da informação que cada interessado deve possuir do requisito. Em contrapartida, requisitos não devem contemplar informações de gestão do projeto, detalhes da arquitetura, projeto ou implementação, bem como informações sobre como o sistema será validado.

Os requisitos de *software* devem dar ênfase somente no comportamento do *software* que será construído e mantido.

Além da classificação em níveis, os requisitos também são categorizados em: requisitos funcionais e requisitos não funcionais.

3.2.1 Requisitos funcionais

Requisitos funcionais (RF) descrevem o comportamento esperado de um sistema de *software*, explicita o que o sistema deve fazer e idealmente o que o sistema não deve fazer (SOMMERVILLE, 2010).



Observação

É uma boa prática descrever o que o sistema não deve fazer. Descrevemos o que está no escopo e, principalmente, o que está fora dele com o objetivo de diminuir possíveis ambiguidades na interpretação.

Pfleeger (2004) define requisitos funcionais, como o detalhamento da interação entre o sistema de *software* e o seu ambiente. Podemos considerar que um componente importante do ambiente do *software* é o seu usuário. Logo, podemos considerar que o requisito funcional é uma descrição detalhada da interação entre o *software* e o usuário.

No exemplo do cliente de autoatendimento bancário, são exemplos de requisitos funcionais:

- O cliente deve ser capaz de verificar seu saldo de conta-corrente na tela do terminal.
- O cliente deve ser capaz de verificar quais tipos de notas estão disponíveis para saque.
- O cliente deve ser informado corretamente quando as cédulas forem dispensadas.
- O cliente deve ser capaz de efetuar um saque, em dinheiro, de sua conta-corrente, desde que ele possua saldo e/ou limite para saque.

3.2.2 Requisitos não funcionais

Requisitos não funcionais (RNF) descrevem restrições sobre os serviços oferecidos pelo sistema de *software* (SOMMERVILLE, 2010).

Os requisitos funcionais são insuficientes para descrever o sistema de *software*, pois é necessário descrever outros aspectos: atributos do sistema e atributos do ambiente do sistema, normalmente classificados como requisitos não funcionais.

Existem várias propostas para classificação de requisitos não funcionais, Sommerville (2010), por exemplo, classifica-os de acordo com o representado na figura a seguir.

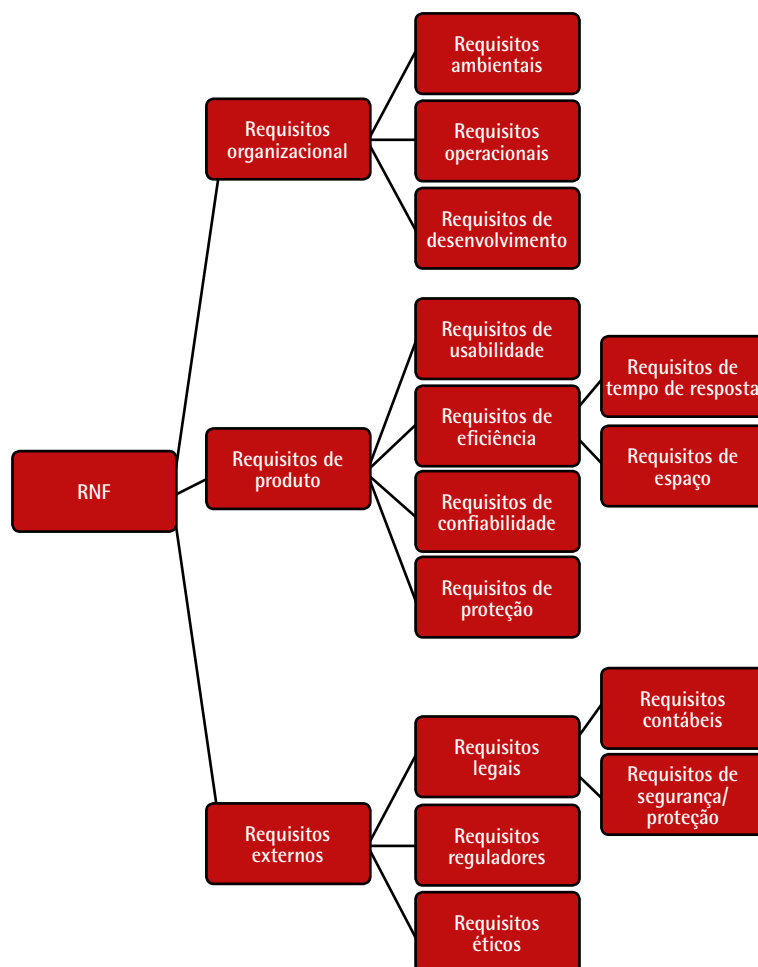


Figura 11 – Requisitos não funcionais, segundo a classificação de Sommerville (2010, p. 88)

Essa classificação é feita a partir da origem dos requisitos não funcionais, que são:

- Requisitos Organizacionais: provenientes da organização do cliente, como a padronização de uma linguagem de desenvolvimento.
- Requisitos de Produto: são aqueles que restringem o comportamento do sistema de *software*, como o espaço em disco que ele irá ocupar.

- Requisitos Externos: são requisitos de fora da fronteira do sistema de *software*, como requisitos legais, de cumprimento da legislação.



Saiba mais

Existem várias propostas de classificação dos RNFs. Dentre as quais a "Árvore de Características de Qualidade de *Software*", que define um conjunto mínimo de padrões de qualidade que um *software* deve apresentar e pode ser visto em BOEHM, B.; BROWN, J. R.; LIPOW, M. Quantitative Evaluation of Software Quality. In: *Procedures of 2nd International Conference of Soft. Eng. San Francisco*. 1976. p 9.

Além da proposta de Sommerville (2010), a norma ISO/IEC 25010:2011 trata da classificação e da definição de requisitos não funcionais.



Observação

O International Organization for Standardization (ISSO) é um grupo fundado em 1947, sem relações com órgãos governamentais, localizado na cidade de Genebra, na Suíça, com o objetivo de definir diretrizes normativas.

A norma descreve seis características que definem a qualidade de *software*: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. Essas características, também denominadas atributos de qualidade, são comumente usadas quando trabalhamos com requisitos não funcionais.

O quadro a seguir mostra a definição dos atributos de qualidade de acordo com a ISO 25010 (2011).

Quadro 6 – Atributos de qualidade segundo a ISO 25010 (2011)

Atributo	Descrição
Funcionalidade	Está ligado à capacidade do sistema de <i>software</i> de prover funcionalidades que atendam às necessidades explícitas e implícitas quando usado sob as condições especificadas.
Confiabilidade	Está ligado à capacidade do sistema de <i>software</i> de manter um determinado nível de desempenho quando usado sob as condições especificadas.
Usabilidade	Está ligado à capacidade do sistema de <i>software</i> de auxiliar os usuários na realização de suas tarefas, de maneira produtiva (ROCHA; BARANAUSKAS, 2003).
Eficiência	Está ligado à capacidade do sistema de <i>software</i> de prover desempenho apropriado, relativo à quantidade de recursos utilizados.
Manutenibilidade	Está ligado à capacidade do sistema de <i>software</i> de ser modificado, sendo que essa modificação pode ser uma correção, melhoria ou adaptação.
Portabilidade	Está ligado à capacidade do sistema de <i>software</i> de ser portátil entre plataformas de ambientes.

Adaptado de: ISO 25010 (2011, p.16-21).

A figura a seguir mostra a estrutura hierárquica dos atributos de qualidade, quanto à sua classificação, proposta pela norma ISO 25010.

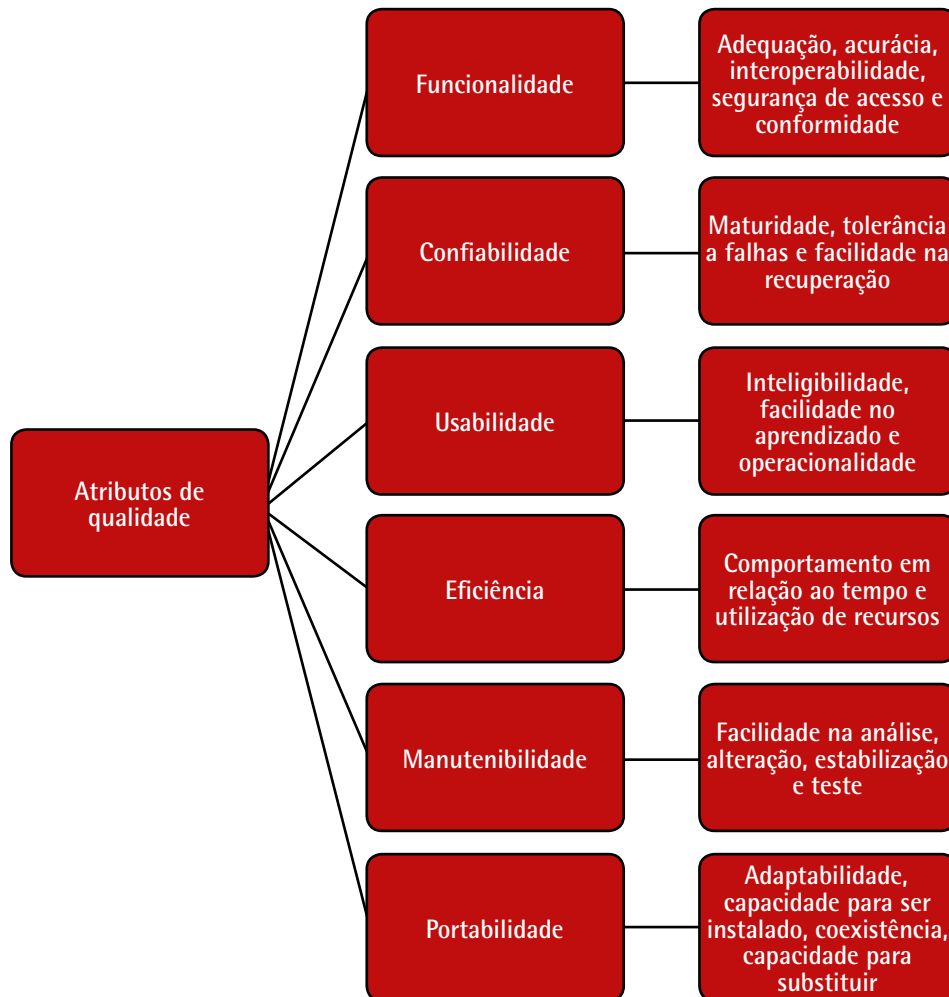


Figura 12 – Requisitos não funcionais, segundo a classificação de ISO 25010 (2011, p. 14)

Conforme ilustrado, os seis atributos de qualidade – funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade – também possuem uma estratificação para outros atributos de qualidade, conforme citado também por Cortês e Chiossi (2001).

- Funcionalidade:
 - Adequação: é a garantia de que o *software* possui as funções que foram definidas.
 - Acurácia: está relacionado ao grau de precisão dos resultados gerados pelo *software*.
 - Interoperabilidade: é a capacidade do *software* de interagir com outros sistemas.
 - Segurança de acesso: está ligado à capacidade do *software* de prevenir acessos por pessoas ou aplicações não autorizados.

- Conformidade: se o *software* foi desenvolvido seguindo os padrões, convenções ou regras pré-estabelecidas no projeto.
- Confiabilidade:
 - Maturidade: está relacionada à baixa frequência de falhas ou defeitos do *software* em operação.
 - Tolerância a falhas: está relacionada a um determinado nível de desempenho que o *software* deve atingir, mesmo em momentos de problemas.
 - Facilidade na recuperação: está ligada à capacidade do *software* de se restabelecer, restabelecer seus dados e parâmetros sem que haja necessidade de intervenção, após uma falha.
- Usabilidade:
 - Inteligibilidade: medida da facilidade do usuário para reconhecer a lógica de funcionamento do *software*.
 - Facilidade no aprendizado: medida da facilidade encontrada pelo usuário para aprender a utilizar o *software*.
 - Operacionalidade: medida da facilidade para operar o produto com segurança e sem falhas.
- Eficiência:
 - Comportamento em relação ao tempo: está relacionado ao tempo de resposta e de processamento do *software* em seus serviços.
 - Utilização de recursos: está relacionado à quantidade de recursos utilizados (CPU, Memória, Processador) e ao tempo de resposta e de processamento do *software* em seus serviços.
- Manutenibilidade:
 - Facilidade na análise: medida do esforço necessário para diagnosticar falhas e localizar as partes para corrigir os problemas.
 - Facilidade na alteração: medida do esforço necessário para corrigir as falhas ou adequar o produto a eventuais mudanças.
 - Facilidade na estabilização: medida do esforço necessário em se estabilizar o *software* após as alterações.
 - Facilidade no teste: medida do esforço necessário para testar o produto de *software* alterado.

- Portabilidade:
 - Adaptabilidade: está ligado à facilidade em se adaptar o *software* para funcionar em outros ambientes.
 - Capacidade para ser instalado: medida do esforço necessário para instalar o *software* em um ambiente de operação.
 - Coexistência: está relacionado ao nível de conformidade do *software* a padrões referentes à portabilidade.
 - Capacidade para substituir: medida do esforço necessário em substituir um *software* por outro previamente especificado.

Os requisitos não funcionais, em geral, são levantados de maneira informal e abstrata e logo são colocados em segundo plano no processo de elicitação de requisitos. São, em geral, altamente complexos em sua documentação, rastreabilidade e validação.

A qualidade na elicitação dos requisitos não funcionais é fator crítico de sucesso para o projeto de *software*. O fator de sucesso deve-se ao fato de que todo requisito não funcional especificado deve fornecer subsídios numéricos para que seja validado.

Talvez seja essa a maior dificuldade em se definir um requisito não funcional, se compararmos ao requisito funcional. Enquanto requisitos funcionais são facilmente verificáveis, requisitos não funcionais nos parecem, em um primeiro momento, mais abstratos.

É importante que tenhamos em mente que não importa o tipo do requisito, funcional ou não funcional, ele sempre deve ser verificável e validado (BOURQUE; FAIRLEY, 2004).

Por exemplo, em um sistema de autoatendimento bancário, é desejável que tenhamos disponibilidade, ou seja, um equipamento deve estar disponível a qualquer momento que um cliente deseje efetuar um saque.

No entanto, para descrevermos um requisito não funcional, somente o desejo não basta, precisamos de características, dados e números que nos possibilitem garantir que a disponibilidade será atingida ou não.

Se especificarmos que o sistema deve estar disponível por 99,5% do tempo nos dias úteis, das 6h às 22h, temos subsídios suficientes para mensurar o atingimento ou não dessa meta de qualidade.



Saiba mais

A qualidade de *software* é disciplina extensa e importante para ser debatida apenas no contexto deste livro-texto. Veja no trabalho de Guerra e Colombo (2009) como a norma ISO 9126 pode ser aplicada na avaliação da qualidade:

GUERRA, A. C.; COLOMBO, R. M. T. *Tecnologia da informação: qualidade de produto de software*. Brasília: PBQP, 2009.

3.3 Processo de Engenharia de Requisitos

Como vimos anteriormente, Engenharia de Requisitos é um conjunto de métodos, procedimentos e ferramentas que tem por objetivo descobrir, analisar, documentar, verificar e validar esses requisitos (SOMMERVILLE, 2010).

O processo de engenharia de requisitos tem como objetivo obter requisitos definidos, especificados e modelos de sistema a partir de fontes de requisitos (BOURQUE; FAIRLEY, 2004). As fontes de requisitos podem ser, segundo Kotonya e Sommerville (1998):

- Sistemas de informações existentes.
- Necessidade dos interessados (*stakeholders*).
- Padrões da organização.
- Informações do Domínio.
- Regulamentos (ou legislações).

Logo, temos que o processo de engenharia de requisitos tem como entradas as fontes de requisitos e como saída requisitos e modelos, conforme representado na figura a seguir.

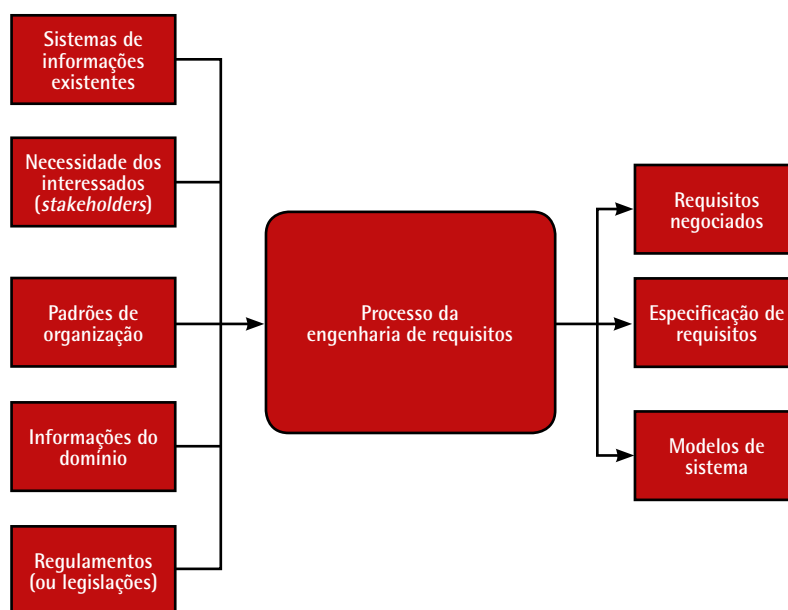


Figura 13 – Visão geral de processo ER

O processo de engenharia de requisitos possui cinco atividades principais: elicitação, análise e negociação, documentação, validação e gerenciamento (KOTONYA; SOMMERVILLE, 1998).

A figura a seguir mostra o modelo geral de processo da engenharia de requisitos:

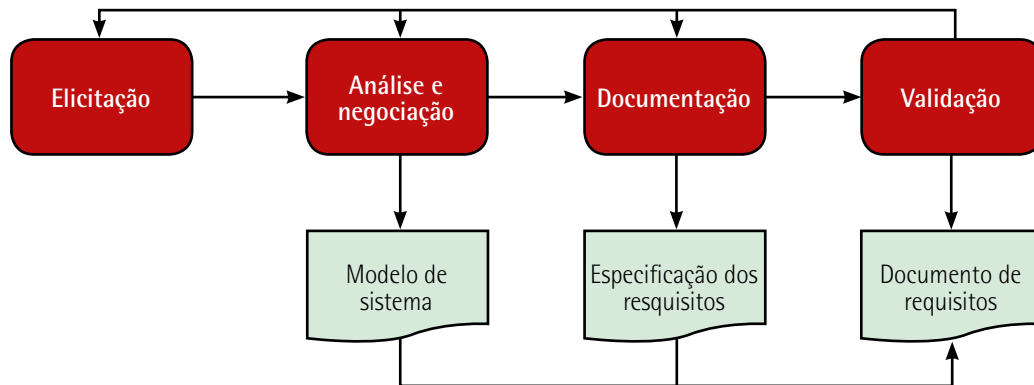


Figura 14 – Modelo geral de processo da ER

- Elicitação: descoberta dos requisitos a partir das fontes de requisitos (sistemas de informação existentes, *stakeholders*, padrões da organização, informações de domínio e regulamentos).
- Análise e negociação: os requisitos são analisados e os conflitos resolvidos por meio da negociação com os interessados. Nessa atividade é produzido o Modelo de Sistema com os requisitos do usuário.
- Documentação: os requisitos são detalhados de tal modo que permitam a realização das próximas atividades do desenvolvimento. É produzida a especificação dos requisitos contendo os requisitos de usuário e de sistema.
- Validação: os requisitos são validados de acordo com os critérios definidos na documentação dos requisitos.
- Gerência: a atividade de gerenciamento de requisitos, como se pode notar, não está representada na figura, isso por que a meta da gerência de requisitos é o controle da mudança dos requisitos ao longo do processo da engenharia de requisitos. Logo, é uma atividade que suporta todo o ciclo de vida do *software*.

3.3.1 Elicitação de requisitos

A qualidade dos requisitos é influenciada diretamente pelas técnicas empregadas durante a elicitação de requisitos (HICKEY; DAVIS, 2002).

A elicitação de requisitos tem como objetivo a descoberta dos requisitos e das fronteiras do sistema, que determinam o escopo do projeto. Nesta fase, é fundamental que haja uma interação e uma cooperação grande entre todos os interessados – clientes, usuários e analistas – com o objetivo de que todas as informações a respeito das necessidades do negócio sejam extraídas. Logo, podemos afirmar que essa é uma atividade de grande dependência do fator humano.

Um dos grandes problemas encontrados na prática e relatado por Hickey e Davis (2003) é que depositamos todas as esperanças de sucesso do levantamento dos requisitos na figura do analista de requisitos, se este tiver uma *expertise* maior no negócio, tanto melhor. Ocorre que apenas isso não basta, assim como não é suficiente sentar juntamente com o usuário e perguntar o que ele deseja. O usuário pode não ter todas as informações necessárias, pode eventualmente não se lembrar de tudo, assim como nós, enquanto analistas, podemos não conseguir extrair ou interpretar todas as informações corretamente.



Saiba mais

Martin Fowler, respeitado engenheiro de *software* e um dos expoentes quando falamos em processos de *software*, tem uma visão interessante sobre a "imprevisibilidade" dos requisitos em:

FOWLER, M. The unpredictability of requirements. In: _____. *The new methodology*. 2005. Disponível em: <<http://www.martinfowler.com/articles/newMethodology.html#TheUnpredictabilityOfRequirements>>. Acesso em: 30 out. 2014.

Para esclarecer essa situação, Cristel e Kung (1992 *apud* PRESSMAN, 2006) citam alguns dos problemas identificados durante o levantamento:

- Problemas de escopo: os limites do sistema não são definidos de forma correta.
- Problemas de entendimento: o usuário pode ter falta de conhecimento amplo do domínio e omitir informações que eventualmente considere desnecessárias ou óbvias, gerando requisitos ambíguos, conflitantes com outros requisitos de outros usuários, impossíveis de serem implementados e testados.
- Problemas de volatilidade: costumamos brincar que a única certeza que temos em requisitos é que eles mudam com o tempo, logo devemos organizar a elicitação dos requisitos de tal forma a nos adequarmos a essa realidade.

Existem muitas técnicas para elicitação de requisitos, trabalharemos aqui as mais difundidas que estão no livro de Kotonya e Sommerville (1998) e no *Swebok*, de Bourque e Fairley (2004).



Observação

As técnicas de elicitação de Kotonya e Sommerville (1998) são utilizadas em muitos cenários e em diferentes processos da Engenharia de Software. Um exemplo disso é o trabalho de Paetsch, Eberlein e Maurer (2003), no qual os autores aplicam a engenharia de requisitos em um modelo ágil de desenvolvimento.

Entrevistas

Técnica mais tradicional e amplamente utilizada. Trata-se basicamente da conversa direcionada a um objetivo: o de obter informações sobre um problema de um entrevistado, na maioria das vezes o próprio usuário.

Cenários

Nesta técnica é montado um cenário de interações entre o usuário e o sistema para que ele execute uma determinada tarefa. O usuário descreve o passo a passo das interações e das atividades que executa para realização de uma tarefa e as informações de que necessita para execução dessas atividades, ou seja, descreve o cenário completo de sua rotina.

Protótipos

Como já dissemos, um protótipo é uma versão não operacional do sistema de *software* que permite, principalmente ao analista, esclarecer eventuais requisitos.

Reuniões facilitadas

Um exemplo dessa técnica muito utilizada é o *brainstorming*, no qual um grupo de usuários é reunido para debater a respeito de um determinado requisito.

É importante que essa reunião ocorra com um mediador, que define o objetivo a ser tratado, capte as principais ideias, ponha-as em debate e não permita que o foco da reunião se perca. A falta de direcionamento e a perda do foco talvez seja a principal causa de insucessos em reuniões dirigidas.

Observação

Os analistas aprendem sobre as tarefas do usuário por imersão em seu e observando como eles interagem com seu *software* e uns com os outros.

Estas técnicas são relativamente custosas, mas no geral são instrutivas, pois ilustram que tarefas do usuário e processos de negócios podem ser muito sutis e complexos, o que invariavelmente dificulta a descrição.

Análise de documentos

Os requisitos são elicitados a partir de documentos da organização. Um exemplo muito próximo à nossa realidade são os editais de licitação para desenvolvimento de *software*, neles, via de regra, estão todos os requisitos desejáveis para o projeto.

É importante que tenhamos sempre em mente que a elicitação de requisitos, qualquer que seja a técnica escolhida, é uma atividade cooperativa.



Saiba mais

Como dissemos, existem muitas técnicas de elicitação. Veja o artigo de Hickey e Davis (2003), no qual os autores apresentam um estudo de caso da aplicação de diversas dessas técnicas em diversos cenários:

HICKEY, A. M.; DAVIS, A. M. Elicitation technique selection: how do experts do it? *In: INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE*, 11., 2003, Monterey. *Proceedings of...* Monterey Bay: IEEE, 2003. p. 169-178.

3.3.2 Análise de requisitos

Com os requisitos identificados, é necessário explicitar as informações obtidas na elicitação. Esse processo auxilia no entendimento dos requisitos elicitados, eliminando possíveis ambiguidades e problemas que possam ter passado despercebidos pelo processo de elicitação.

Como vimos anteriormente, o usuário não tem a obrigação de conhecer o domínio todo do problema, o que pode eventualmente gerar requisitos que conflitem com requisitos de outros usuários. Pelo mesmo motivo, ainda é possível que as necessidades dos usuários sejam conflitantes com as necessidades do cliente.

É comum que na fase de análise dos requisitos elicitados, surjam requisitos conflitantes, esses conflitos precisam ser resolvidos por um processo de negociação. Uma negociação envolve três fases:

- Discussão: os conflitos são apresentados e discutidos entre os interessados.
- Priorização: os requisitos são priorizados sempre levando em consideração os requisitos críticos em detrimento daqueles que levem menor valor ao negócio.
- Concordância: as soluções são identificadas, debatidas e as mudanças necessárias são feitas. Um acordo é feito entre todas as partes interessadas.

Pressman (2006, p. 119) faz uma observação interessante a respeito do processo de negociação: "em uma negociação efetiva não existem ganhadores ou perdedores, ambos os lados ganham, pois existe um trato".

Com os conflitos solucionados, não devemos perder a ênfase da análise de requisitos, que é explicitar as informações obtidas na elicitação dos requisitos.

Explicitar os requisitos significa especificar as características operacionais do *software*, descrever suas restrições e suas interações com outros sistemas a fim de estabelecer um modelo conceitual do problema (PFLEEGER, 2004).

Um modelo pode ser considerado como a representação de uma realidade e pode ser utilizado de diversas maneiras. No caso do modelo de requisitos, também chamado de modelo conceitual, estamos preocupados em representar o domínio do problema.

Nessa fase de modelagem ainda não estamos preocupados em dar a solução para um problema, mas sim mapeá-lo com precisão.

Os modelos produzidos na análise de requisitos são importantes, pois (PRESSMAN, 2006):

- Sistematizam e facilitam a análise de requisitos, pois ajudam o analista e os interessados a entenderem a função e o comportamento do *software*.
- São chaves para a validação da especificação.

3.3.3 Documentação de requisitos

A documentação dos requisitos tem por objetivo formalizar os requisitos e modelos definidos nas etapas anteriores (BOURQUE; FAIRLEY, 2004).

Mais do que isso, documentos têm a finalidade de comunicar. Nesse ponto, devemos estar atentos aos diferentes papéis de interessados em cada tipo de requisito.

Como vimos, requisitos de usuário são utilizados por clientes, usuários, gerentes de projeto e arquitetos de sistema, pois definem em alto nível as necessidades, o que define, dentre outras coisas, o escopo do projeto.

Requisitos de sistemas são utilizados por usuários, arquitetos de sistema e desenvolvedores, pois podem definir uma sequência de implementação o que influencia diretamente na solução dada.

Temos diferentes interessados com necessidades e pontos de vistas diferentes. O Swebok (BOURQUE; FAIRLEY, 2004) sugere três documentos de visão:

- Documento de definição de sistema: este documento contém os requisitos de sistema, ou seja, requisitos em alto nível de abstração na perspectiva do domínio. Podendo incluir requisitos não funcionais e modelos conceituais.
- Especificação de requisitos de sistema: este documento contém os requisitos na visão sistêmica, ou seja, não somente os requisitos do sistema de *software*, mas tudo que o circunda.
- Especificação de requisitos de *software*: são os requisitos na visão do sistema de *software*. Esse documento pode ser utilizado como base para o desenvolvimento e para as futuras validações.



Lembrete

O Software Engineering Body of Knowledge – Swebok (BOURQUE; FAIRLEY, 2004) está disponível no site do IEEE e é material de apoio e leitura importante para engenheiros de *software*.

Existem diversas abordagens para documentação de requisitos, Pfleeger (2004), por exemplo, aborda a criação de dois tipos de documentos: um Documento de Definição de Requisitos e um Documento de Especificação de Requisitos.

Qualquer que seja a abordagem escolhida, um bom documento de requisitos não é ambíguo, é correto, completo e perfeitamente interpretável e, muitas vezes, pode ser utilizado como parte de um contrato com o cliente (PAETSCH; EBERLEIN; MAURER, 2003).

3.3.4 Validação de requisitos

Todos os documentos de requisitos estão sujeitos a passarem por procedimentos de verificação e validação (BOURQUE; FAIRLEY, 2004). O objetivo do processo de validação é assegurar que o trabalho de elicitação, análise e documentação dos requisitos está consistente com o domínio do projeto.

Nesta fase, é comum encontrarmos problemas de inconsistência, ambiguidade, omissão de informações ou informações erradas. Existe uma série de questionamentos que devem ser feitos nesta fase (SOMMERVILLE, 2010):

- Os requisitos descritos são consistentes? Ou seja, não entram em conflito?
- Os requisitos descritos são completos? Ou seja, definem todas as funcionalidades e restrições previstas na elicitação?
- Os requisitos são passíveis de serem implementados?
- Os requisitos são verificáveis? Ou seja, podemos escrever testes que possam verificar que o sistema implementado atende a cada requisito elicitado? Requisitos que não podem ser testados não são requisitos, mas sim apenas desejos (BOURQUE; FAIRLEY, 2004).

Para responder a esses questionamentos, existe uma série de métodos para validação de requisitos. Descreveremos aqui os citados por Sommerville (2010), Kotonya e Sommerville (1998) e pelo Swebok (BOURQUE; FAIRLEY, 2004).

- Revisão de Requisitos: os requisitos são avaliados por uma equipe de revisores, de preferência com pouca participação no processo de engenharia de requisitos, com o objetivo de verificar possíveis inconsistências e erros.

- Prototipação: como debatido anteriormente, é construído um modelo executável do sistema.
- Testes de aceitação: podemos escrever testes que possam verificar que o sistema implementado atende a cada requisito elicitado. Temos que ter a clara noção e repassar essa visão aos usuários de que requisitos que não podem ser testados não são requisitos, mas sim desejos, não devendo assim, tomarem espaço no escopo do projeto.



Lembrete

Lembre-se que a garantia de qualidade de *software* é disciplina extensa e que não devemos subestimar essa fase.



Saiba mais

Saiba mais sobre o processo de validação de requisitos e da garantia da qualidade em:

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. *Qualidade de software: teoria e prática*. São Paulo: Prentice Hall, 2001.

3.3.5 Gerenciamento de requisitos

Na engenharia de requisitos, devemos nos acostumar com a idéia de que requisitos podem (e quase sempre) mudam. Inevitavelmente, requisitos podem ser incompletos e inconsistente, bem como novos requisitos podem surgir, e comumente surgem de algumas situações como: uma melhor compreensão do sistema em desenvolvimento e mudanças das necessidades do negócio originadas pelo mercado, por mudanças internas ou mudanças legais, essa última, tão comum à realidade brasileira.

A meta da gerência de requisitos é o controle da mudança dos requisitos ao longo do processo de engenharia de requisitos (PRESSMAN, 2006). Controlar mudanças significa, entre outras coisas, analisar o impacto dessas mudanças.

Conforme vimos anteriormente, requisitos podem ser conflitantes, bem como podemos ter requisitos dependentes. Diante desse cenário, como identificarmos o risco da alteração de um requisito, sem que tenhamos mapeados quais requisitos são dependentes uns dos outros?

Por isso, Kotonya e Sommerville (1998) indicam que, além de controlar a mudança dos requisitos, a gerência de requisitos tem mais dois objetivos:

- Gerenciar o relacionamento entre requisitos.

- Gerenciar o relacionamento entre requisitos e os documentos produzidos durante todo o projeto.

Uma boa gestão de requisitos passa principalmente pela definição de uma política organizacional, que muitas vezes é impactada pelo nível de maturidade dessa organização.

Para definição de uma política organizacional de produção de *software* com qualidade, existem alguns modelos de processo que definem de maneira completa e complexa todas as atividades, artefatos a serem produzidos.

Dois dos modelos mais difundidos atualmente são: CMMI (Capability Maturity Model for Software) e o MPS.BR (Melhoria de Processo do Software Brasileiro). Em ambos os modelos, o processo de gerenciamento é tratado como um processo de apoio ao desenvolvimento.

O assunto processo de gerenciamento é amplo e foge da temática deste livro-texto, vamos detalhar aqui apenas duas das atividades fundamentais no processo de gestão de requisitos, citadas tanto nos modelos CMMI e MPS.BR quanto por Wiegers (2003), que são: controle de versão e rastreabilidade.



Saiba mais

Para mais informações a respeito do modelo MPS, sua estrutura e aplicabilidade, visite o site:

<<http://www.softex.br/mpsbr/guias/>>.

No site da empresa ISD Brasil, podemos obter maiores detalhes a respeito do modelo CMMI. Acesse:

ISD BRASIL. *Perguntas frequentes: o que é CMMI-DEV?* Disponível em: <<http://www.isdbrasil.com.br/o-que-e-cmmi-dev.php>>. Acesso em: 18 dez. 2014.

Controle de versão

Já vimos que um documento de requisitos e os documentos que o compõem são considerados a base para o desenvolvimento. Essa base pode ser considerada uma versão.

É importante que tenhamos sob controle, por qualquer mecanismo que seja, quais documentos fazem parte de uma versão base, uma vez que, a partir da mudança de um requisito, é gerada uma nova versão.

Sem um controle de versão, seria praticamente impossível garantir o trabalho em equipe e manter o histórico de evolução do *software*.

Rastreabilidade

Rastreabilidade especifica as dependências de um requisito para com outro e de um requisito com cada artefato gerado ao longo do projeto. Quando falamos de artefatos, não estamos falando apenas dos gerados durante a engenharia de requisitos, mas sim em todo o projeto, como qual requisito está relacionado à qual solução de *software* (SOMMERVILLE, 2010).

Segundo Palmer (1997), é uma trilha de auditoria que nos permite identificar tudo que pode ser afetado no caso da mudança de um requisito, daí a necessidade em termos um controle sobre a dependência entre requisitos e da ligação entre requisitos e artefatos produzidos.

Esse controle pode ser feito utilizando uma matriz de rastreabilidade, que tem por objetivo cruzar um requisito com outro dependente ou um requisito com um artefato.

Existem diversas matrizes de requisitos. Entre as mais utilizadas, podemos destacar (KOTONYA; SOMMERVILLE, 1998):

- Matriz que indica o relacionamento requisito x requisito.
- Matriz que indica o relacionamento requisito x fonte do requisito.
- Matriz que indica o relacionamento requisito x subsistemas, que descreve quais subsistemas atendem àquele requisito.
- Matriz que indica o relacionamento entre um requisito elicitado e um artefato produzido no processo da Engenharia de Software, como um caso de teste ou um documento de requisitos.

A figura a seguir mostra um exemplo de matriz de rastreabilidade que indica a dependência entre os requisitos funcionais e não funcionais.

Depende de	Requisito funcional 1 – usabilidade	Requisito funcional 2 – usabilidade	Requisito funcional 3 – confiabilidade
Requisito funcional X	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Requisito funcional Y	<input checked="" type="checkbox"/>		
Requisito funcional Z			<input checked="" type="checkbox"/>
Requisito funcional W		<input checked="" type="checkbox"/>	

Figura 15 – Matriz de Rastreabilidade

4 MODELAGEM DE CASOS DE USO

Vimos que a engenharia de requisitos é um conjunto de métodos, procedimentos e ferramentas que tem por objetivo descobrir, analisar, documentar, verificar e validar esses requisitos. Vimos

ainda uma visão de processo contendo as principais atividades a serem executadas na engenharia de requisitos, suas principais características e objetivos e uma visão a respeito da sequência de execução dessas atividades.

Como se pode notar, detalhamos os métodos e ferramentas utilizados em uma abordagem para elicitação de requisitos, citamos algumas das principais técnicas e como elicitar os requisitos de sistema e de usuário em alto nível, tal como é o objetivo da fase de levantamento de requisitos.

Logo, é possível notar que conseguimos fechar um ciclo: processo, ferramenta e método, no que diz respeito à elicitação de requisitos.

O objetivo agora é apresentar a modelagem de caso de uso como uma abordagem que combina método e ferramenta e que cumpra com os objetivos a serem atingidos na fase de análise de requisitos.

A análise de requisitos tem como objetivo explicitar os requisitos elicitados, o que significa especificar as características operacionais do *software*, descrever suas restrições e suas interações com outros sistemas a fim de estabelecer um modelo conceitual do problema (PFLEEGER, 2004).

Lembremos ainda que esse modelo conceitual pode ser interpretado como uma "maquete da realidade", utilizada para representar o domínio do problema. Além disso, esses modelos são utilizados na comunicação entre clientes, arquiteto, gerentes de projeto e desenvolvedores.

Segundo Bezerra (2006, p. 45), "modelo de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que interagem com ele"

A definição do professor Eduardo Bezerra, corrobora com a visão dos criadores da modelagem de casos de uso, que diz que não existem sistemas de *software* que existam isoladamente (BOOCH; JACOBSON; RUMBAUGH, 2006). Todo sistema de *software* interage com algum agente externo a ele, seja um usuário ou outro sistema, com o objetivo de realizar uma determinada ação.

Um caso de uso é a descrição de uma determinada ação ou comportamento de um sistema que produz um resultado para um determinado agente externo a esse sistema, denominado ator.



Observação

Muito embora o conceito de modelagem de casos de uso esteja extremamente ligado ao conceito de UML na literatura, até mesmo por terem os mesmos criadores (Booch, Jacobson e Rumbaugh), devemos dissociar os conceitos. Modelagem de caso de uso é um método, aplicável a qualquer modelo de processo de engenharia de requisitos, e UML é uma ferramenta de apoio.

4.1 Casos de uso e atores

Caso de uso é a descrição de uma sequência de atividades executadas por um agente externo ao sistema sem que sejam revelados detalhes do funcionamento interno ao sistema, por isso dizemos que o caso de uso mostra a visão comportamental externa ao sistema (BEZERRA, 2006).

Se bem descritos e definidos, casos de uso definem um denominador comum, de conhecimento do domínio do problema e das funcionalidades do sistema, que pode ser interpretado facilmente por usuários, analistas e desenvolvedores (BOOCH; JACOBSON; RUMBAUGH, 2006).

Atores são os agentes externos ao sistema que executam uma determinada ação e que esperam algum resultado, ou seja, interagem diretamente com o sistema a partir dos casos de uso.

Algumas considerações importantes a respeito de atores:

- Ator é qualquer entidade externa que interage com o sistema: usuários, outros sistemas de *software* ou até mesmo dispositivos de *hardware*.
- Um ator nunca é um componente interno do sistema, ou mesmo qualquer detalhe interno ou componente de implementação dele.
- Um sistema nunca é um ator de si mesmo.

É uma boa prática de projeto identificar o caso de uso por um nome que o represente, costuma-se utilizar frases iniciadas com verbos no infinitivo, seguido pela meta ou tarefa a ser realizada. Por exemplo: "Efetuar saque de conta-corrente".

Enquanto para identificar um ator, costuma-se utilizar um substantivo, sempre no singular. Evite tipos muito genéricos, como Clientes ou Usuários. Opte sempre por uma descrição próxima à realidade, por exemplo: Cliente Pessoa Física ou Cliente Pessoa Jurídica.

4.2 Descrição de casos de uso

Muitas são as discussões na literatura a respeito do nível de detalhamento necessário para se descrever um caso de uso. Podemos considerar a descrição em linguagem natural, desde que sequencial, como uma descrição de caso de uso, conforme mostra o quadro a seguir, no nosso exemplo do autoatendimento.

Quadro 7 – Exemplo de descrição de caso em descritiva natural

Caso de uso: efetuar saque conta-corrente

O cliente insere o cartão. O sistema solicita que o cliente informe a senha. O cliente informa a senha. O sistema exibe as operações possíveis de serem feitas no terminal. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada. O sistema dispensa as cédulas. O cliente retira as notas.



Observação

Note que em momento algum na descrição do caso de uso são usados termos que remetam a soluções técnicas, detalhes de implementação ou aspectos internos do sistema.

Podemos descrever também o passo a passo das atividades de forma numerada, conforme o quadro a seguir, que nada mais é que a mesma descrição exibida no quadro anterior, mas com uma sequência numerada. Veja:

Quadro 8 – Exemplo de descrição de caso em descritiva numerada

Caso de uso: efetuar saque conta-corrente
1. O cliente insere o cartão. 2. O sistema solicita que o cliente informe a senha. 3. O cliente informa a senha. 4. O sistema exibe as operações possíveis de serem feitas no terminal. 5. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada. 6. O sistema dispensa as cédulas. 7. O cliente retira as notas.



Observação

Note que um caso de uso representa uma transação completa, ou seja, com começo, meio e fim.

Repare que, nas descrições dos exemplos anteriores, embora completa sob o ponto de vista da descrição do caso de uso, se tornam incompletas se olharmos sob o ponto de vista da necessidade de completude dos requisitos, que abordamos nas seções anteriores.

Um requisito deve ser completo. Voltando ao nosso exemplo, onde está descrito o comportamento do sistema ou do cliente, caso o cliente não possua saldo em conta corrente, não informe a senha correta ou ainda não possuam notas disponíveis no terminal?

Pensando na completude do modelo, Cockburn (2005) propõe um modelo de descrição de caso de uso contendo alguns elementos que nos guiam a especificar um caso de uso completo.

O quadro a seguir mostra um exemplo de modelo para descrição de caso de uso e, logo abaixo, a explicação de cada elemento.

Quadro 9 – Modelo de descrição de caso de uso

Identificação: efetuar saque conta-corrente.

Escopo: terminal autoatendimento.

Descrição do propósito: esse caso de uso permite ao cliente efetuar um saque de conta-corrente em um terminal de autoatendimento.

Ator primário: cliente.

Interessados: cliente e banco.

Pré-condições: o terminal de autoatendimento deve estar operacional.

Pós-condições: o cliente efetua o saque, a quantia é debitada de sua conta-corrente, as notas são dispensadas e retiradas pelo cliente.

Fluxo normal:

O cliente insere o cartão. O sistema solicita que o cliente informe a senha. O cliente informa a senha. O sistema exibe as operações possíveis de serem feitas no terminal. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada. O sistema dispensa as cédulas. O cliente retira as notas.

Fluxo alternativo:

- Caso o cliente não insira o cartão corretamente, uma mensagem é exibida.
- Caso o cliente não informe a senha corretamente, uma mensagem é exibida e caso informe a senha de forma incorreta por três tentativas seguidas, o cartão será bloqueado e uma mensagem exibida.
- Caso não existam cédulas suficientes ou múltiplas da quantia desejada, uma mensagem é exibida.
- Caso o cliente não possua saldo em conta-corrente, uma mensagem é exibida.

Requisitos relacionados: RF 01 – Efetuar saque; RNF 01 – Disponibilidade de Terminal.

Adaptado de: Cockburn (2005, p. 122).

- **Identificação:** nome do caso de uso.
- **Escopo:** descreve sucintamente a que contexto se refere o caso de uso, podendo ser um processo de negócio, um sistema ou um subsistema.
- **Descrição do propósito:** descrição sucinta do objetivo do caso de uso.
- **Ator primário:** ator principal do caso de uso.
- **Pré-condições e pós-condições:** o que deve ser verdadeiro antes e após a finalização do caso de uso.
- **Fluxo normal:** descrição do passo a passo normal do caso de uso.
- **Fluxo alternativo:** descrição de passos alternativos que podem ser executados dentro de uma sequência normal de passos.
- **Requisitos relacionados:** lista de requisitos que estão representados pelo caso de uso. Um caso de uso pode conectar um ou mais requisitos de tipos diferentes, funcionais e não funcionais.

Outro exemplo aplicado ao modelo de Cockburn (2005) pode ser visto no quadro a seguir, no qual utilizamos a narrativa numerada, o que é interessante, pois conseguimos visualizar exatamente onde entram os passos alternativos no fluxo principal.

Quadro 10 – Modelo de descrição de caso de uso com descritiva numérica

Identificação: efetuar saque conta-corrente.

Escopo: terminal autoatendimento.

Descrição do propósito: esse caso de uso permite ao cliente efetuar um saque de conta-corrente em um terminal de autoatendimento.

Ator primário: cliente.

Interessados: cliente e banco.

Pré-condições: o terminal de autoatendimento deve estar operacional.

Pós-condições: o cliente efetua o saque, a quantia é debitada de sua conta-corrente, as notas são dispensadas e retiradas pelo cliente.

Fluxo normal:

1. O cliente insere o cartão.
2. O sistema solicita que o cliente informe a senha.
3. O cliente informa a senha.
4. O sistema exibe as operações possíveis de serem feitas no terminal.
5. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada.
6. O sistema dispensa as cédulas.
7. O cliente retira as notas.

Fluxo alternativo:

- 1.1 Caso o cliente não insira o cartão corretamente, uma mensagem é exibida.
- 2.1 Caso o cliente não informe a senha corretamente, uma mensagem é exibida.
- 2.2 Caso informe a senha de forma incorreta por três tentativas seguidas, o cartão será bloqueado e uma mensagem exibida.
- 5.1 Caso não existam cédulas suficientes ou múltiplas da quantia desejada, uma mensagem é exibida.
- 5.2 Caso o cliente não possua saldo em conta-corrente, uma mensagem é exibida.

Requisitos relacionados: RF 01 – Efetuar saque; RNF 01 – Disponibilidade de Terminal.

Adaptado de: Cockburn (2005, p. 122).



Saiba mais

Cenários de uso são conjuntos de informações relacionadas ao estímulo provocado pelo usuário, resposta do sistema e condições de ambiente no qual se dá o processamento desse estímulo. Um cenário de uso é a execução específica de um caso de uso. Saiba mais em:

CARROL, J. M. *Scenario-based design: envisioning work and technology in system development*. New York: John Wiley, 1995.

4.3 Diagrama de caso de uso

Somente neste ponto da engenharia de requisitos é que entramos no assunto UML.

Diagrama de casos de uso é um diagrama da UML que tem por objetivo mostrar, a partir de um ponto de vista estático, o conjunto de casos de uso, atores e seus relacionamentos (BOOCH; JACOBSON; RUMBAUGH, 2006).

Visão estática é o termo usado para descrever uma parte do sistema sob o ponto de vista estrutural.



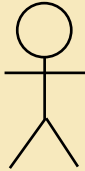

Lembrete

A UML é uma linguagem de modelagem visual de sistemas orientado a objetos, que possui elementos de modelagem gráficos que representam visões de um sistema de *software* e que possuem regras de sintaxe e semântica.

O diagrama de caso de uso representa, em um modelo gráfico, os elementos fundamentais da modelagem de caso de uso: atores e casos de uso.

Os elementos que compõe o diagrama de caso de uso são descritos no quadro a seguir:

Quadro 11 – Notação UML para representação do diagrama de caso de uso

Elemento	Notação UML
Ator	
Caso de Uso	
Fronteira	É a representação da fronteira do diagrama, representado por um retângulo que envolve todos os casos de uso. Corresponde ao escopo da descrição de caso de uso.

A figura a seguir mostra todos os elementos de um diagrama de caso de uso. A linha contínua que liga o ator ao caso de uso representa que aquele ator executa aquele caso de uso, ou seja, que o ator está relacionado ao caso de uso ou podemos encontrar na literatura a terminologia "O Ator executa o Caso de Uso".

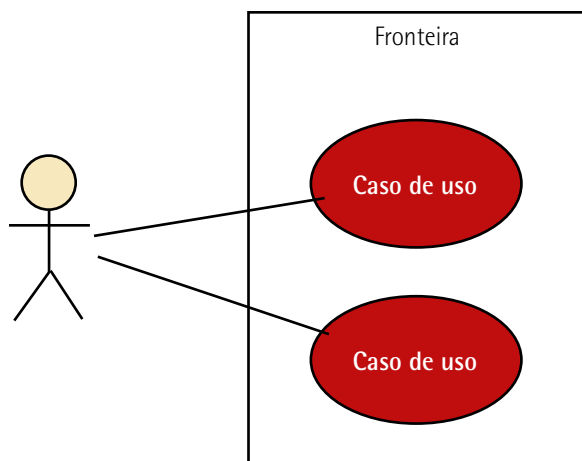


Figura 16 – Diagrama de Caso de Uso

Já a próxima figura mostra um exemplo de um diagrama de caso de uso, voltado para o nosso exemplo do terminal de autoatendimento. Note que acrescentamos mais um ator, o Apoio Operacional, e mais dois casos de uso, Contratar Empréstimo Pessoal e Abastecer Terminal. Veja também que as descrições dos casos de uso seguem as regras descritas anteriormente.

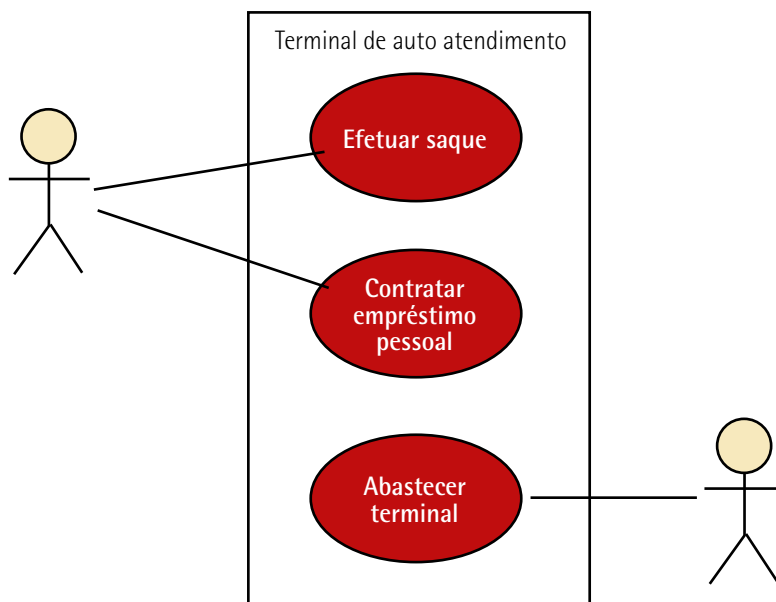


Figura 17 – Exemplo de Diagrama de Caso de Uso

4.4 Estrutura do diagrama de caso de uso

4.4.1 Relacionamento entre casos de uso

Com o objetivo de produzir modelos mais ricos e inteligíveis e fornecer melhores recursos de modelagem, podemos utilizar três tipos de relacionamento entre casos de uso: inclusão, extensão e herança.

Inclusão

Significa que o comportamento definido no caso de uso de inclusão é incorporado ao comportamento do caso de uso base, ou seja, para que este seja executado, obrigatoriamente o caso de uso de inclusão também deverá ser executado (BOOCH; JACOBSON; RUMBAUGH, 2006).

Na notação da UML, um relacionamento de inclusão entre casos de uso é mostrado como uma dependência (seta pontilhada) estereotipada com a palavra-chave **include**, conforme exibido a seguir:

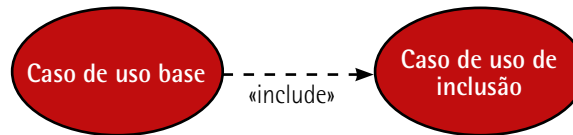


Figura 18 – Notação de inclusão na UML

Por exemplo, na figura a seguir, podemos interpretar que, para o cliente efetuar um saque, é necessário que seja validada a segurança de acesso. Note ainda que não estamos entrando no detalhe de como é feita essa validação, apenas que essa validação é realizada por outro Ator, um sistema externo denominado **Sistema de Segurança**.

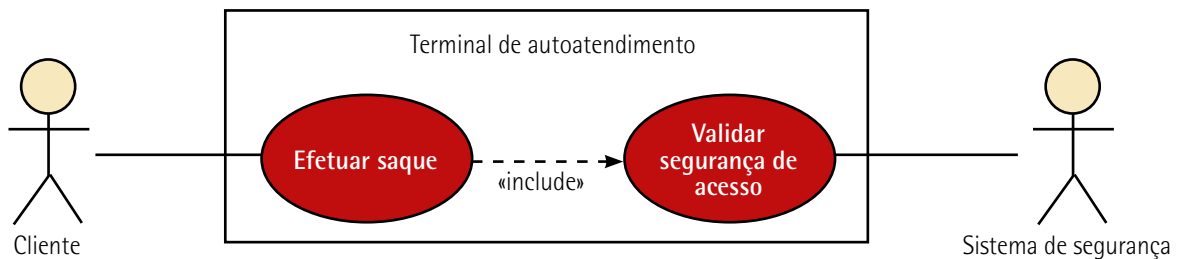


Figura 19 – Exemplo de inclusão na UML

Extensão

Significa que o comportamento definido no caso de uso de extensão pode ou não ser incorporado ao comportamento do caso de uso base, ou seja, para que este seja executado, o caso de uso de extensão pode ou não ser executado (BOOCH; JACOBSON; RUMBAUGH, 2006).

Na notação da UML, um relacionamento de extensão entre casos de uso é mostrado como uma dependência (seta pontilhada) estereotipada com a palavra-chave **extend**, conforme exibido a seguir:

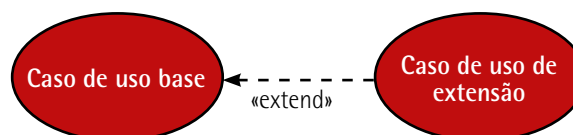


Figura 20 – Notação de extensão na UML

Por exemplo, na figura a seguir, podemos interpretar que o cliente pode solicitar um empréstimo pessoal durante uma operação de saque, o que quer dizer que existe uma condição para execução, ou não, do caso de uso "solicitar empréstimo pessoal".

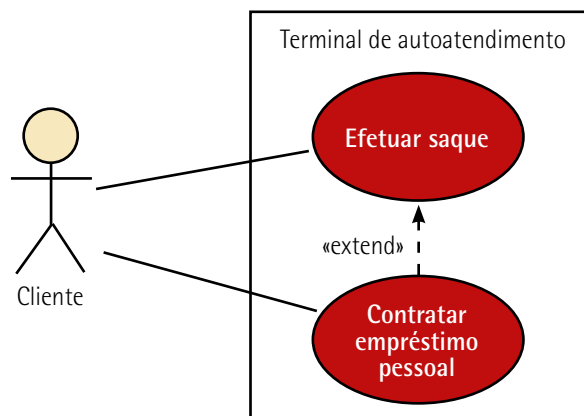


Figura 21 – Exemplo de extensão na UML

Herança

Significa que o caso de uso filho herda o comportamento e o significado do caso de uso pai, acrescentando ou mudando seu comportamento (BOOCH; JACOBSON; RUMBAUGH, 2006).

Na notação da UML, um relacionamento de herança entre casos de uso é mostrado com uma linha cheia com uma seta aberta, conforme exibido a seguir:

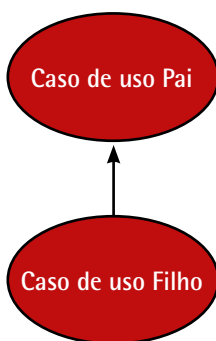


Figura 22 – Notação de herança na UML

Ainda no exemplo da figura, podemos dizer que o caso de uso filho é uma especialização do caso de uso pai, ou que o caso de uso pai é uma generalização do caso de uso filho.

Na figura a seguir, por exemplo, podemos interpretar que a validação de acesso por biometria ou por senha são especializações da validação de segurança de acesso. Os casos de uso filho: "Validar Segurança Acesso por Senha" e "Validar Segurança Acesso por Biometria" possuem exatamente o mesmo comportamento do caso de uso "Validar Segurança Acesso" pai, todavia, possuem comportamentos diferentes.

Podemos interpretar que o objetivo do caso de uso "Validar Segurança Acesso" é certificar que a pessoa no terminal é efetivamente o cliente. Logo, podemos concluir que os casos de uso filho são especializações se, e somente se, mantiverem o mesmo objetivo, não importa por qual meio é feita essa certificação.

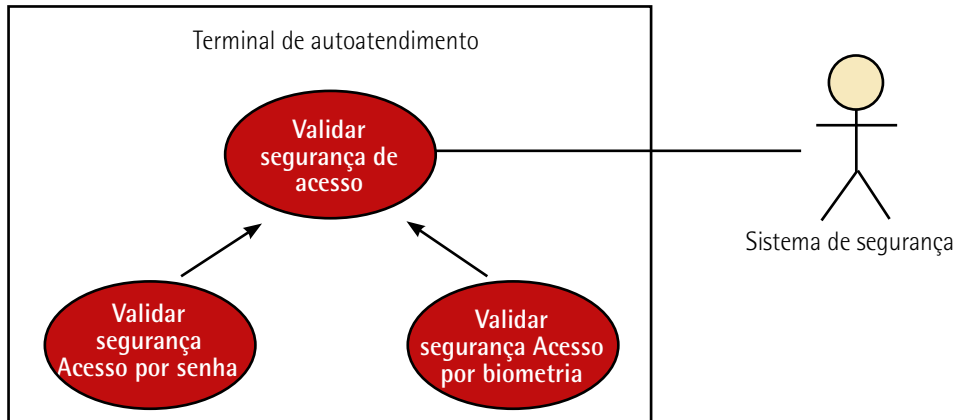


Figura 23 – Exemplo de herança na UML

Note que o uso de qualquer um dos três recursos de relacionamento entre casos de uso reduz sensivelmente a redundância de casos de uso em um modelo.

4.4.2 Relacionamento entre atores

Diferentemente dos casos de uso, atores possuem apenas um tipo de relacionamento: herança.

Significa que o ator filho herda o comportamento e o significado do ator pai, acrescentando ou mudando seu comportamento. Em suma, o ator filho pode executar todos os casos de uso do ator pai e mais aqueles que apenas ele executa (BOOCH; JACOBSON; RUMBAUGH, 2006).

Na notação da UML, um relacionamento de herança entre atores é a mesma notação utilizada para cenários de uso: uma linha cheia com uma seta aberta, conforme exibido a seguir:



Figura 24 – Notação de herança entre atores UML

Neste caso, podemos dizer que o Ator Filho é uma especialização do Ator Pai, ou que o Ator Pai é uma generalização do Ator Filho.

Na figura a seguir, por exemplo, podemos interpretar que o Cliente Pessoa Jurídica pode "Efetuar Saque", "Contratar Empréstimo Pessoal" e "Aumentar Limite de Crédito". Podemos afirmar que o ator Cliente não executa o caso de uso "Aumentar Limite de Crédito", mas somente "Efetuar Saque" e "Contratar Empréstimo Pessoal".

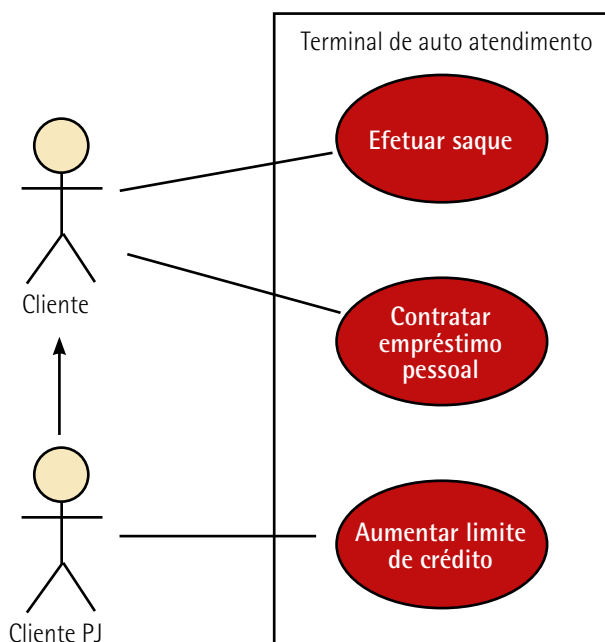


Figura 25 – Exemplo de herança entre atores UML

Na descrição dos casos de uso, que debatemos anteriormente, é importante que indiquemos os pontos de inclusão e extensão, conforme os exemplos dos quadros a seguir:

Quadro 12 – Modelo de descrição de caso de uso com inclusão

Identificação: efetuar saque conta-corrente.

Escopo: terminal autoatendimento.

Descrição do propósito: esse caso de uso permite ao cliente efetuar um saque de conta-corrente em um terminal de autoatendimento.

Ator primário: cliente.

Interessados: cliente e banco.

Pré-condições: o terminal de autoatendimento deve estar operacional.

Pós-condições: o cliente efetua o saque, a quantia é debitada de sua conta-corrente, as notas são dispensadas e retiradas pelo cliente.

Fluxo normal:

1. O cliente insere o cartão.
2. O sistema solicita que o cliente informe a senha.
3. Inclusão para Validar Segurança de Acesso
4. O cliente informa a senha.
5. O sistema exibe as operações possíveis de serem feitas no terminal.
6. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada.
7. O sistema dispensa as cédulas.
8. O cliente retira as notas.

Fluxo alternativo:

- 1.2 Caso o cliente não insira o cartão corretamente, uma mensagem é exibida.
- 2.3 Caso o cliente não informe a senha corretamente, uma mensagem é exibida.
- 2.4 Caso informe a senha de forma incorreta por três tentativas seguidas, o cartão será bloqueado e uma mensagem exibida.
- 5.3. Caso não existam cédulas suficientes ou múltiplas da quantia desejada, uma mensagem é exibida.
- 5.4 Caso o cliente não possua saldo em conta-corrente, uma mensagem é exibida.

Requisitos relacionados: RF 01 – Efetuar saque; RNF 01 – Disponibilidade de Terminal.

Quadro 13 – Modelo de descrição de caso de uso com extensão

Identificação: efetuar saque conta-corrente.

Escopo: terminal autoatendimento.

Descrição do propósito: esse caso de uso permite ao cliente efetuar um saque de conta-corrente em um terminal de autoatendimento.

Ator primário: cliente.

Interessados: cliente e banco.

Pré-condições: o terminal de autoatendimento deve estar operacional.

Pós-condições: o cliente efetua o saque, a quantia é debitada de sua conta-corrente, as notas são dispensadas e retiradas pelo cliente.

Fluxo normal:

1. O cliente insere o cartão.
2. O sistema solicita que o cliente informe a senha.
3. O cliente informa a senha.
4. O sistema exibe as operações possíveis de serem feitas no terminal.
5. O cliente seleciona a operação de saque em conta-corrente e informa a quantia desejada.
6. O sistema dispensa as cédulas.
7. O cliente retira as notas.

Fluxo alternativo:

- 1.3 Caso o cliente não insira o cartão corretamente, uma mensagem é exibida.
- 2.5 Caso o cliente não informe a senha corretamente, uma mensagem é exibida.
- 2.6 Caso informe a senha de forma incorreta por três tentativas seguidas, o cartão será bloqueado e uma mensagem exibida.
- 5.5 Caso não existam cédulas suficientes ou múltiplas da quantia desejada, uma mensagem é exibida.
- 5.6 Caso o cliente não possua saldo em conta-corrente, uma mensagem é exibida.
- 5.7 Extensão para Contratar Empréstimo Pessoal

Requisitos relacionados: RF 01 – Efetuar saque; RNF 01 – Disponibilidade de Terminal.



Saiba mais

Casos de uso podem ser utilizados para calcular o tamanho de um sistema de *software* utilizando o processo de estimativa por pontos de caso de uso, ou *Use Case Points*. O processo de estimativa por caso de uso é explorado no trabalho de pesquisa:

TETILA, E. C. Processos de estimativa de *software* com a métrica *use case points*, PMBOK e RUP. 2007. 140 f. Dissertação (Mestrado em Engenharia de Produção)–Universidade Paulista (UNIP), 2007. Disponível em: <http://www3.unip.br/ensino/pos_graduacao/strictosensu/eng_producao/download/eng_evertoncastelaotetila.swf>. Acesso em: 3 nov. 2014.



Resumo

Requisitos, segundo Sommerville (2010), são as necessidades do cliente refletidas em serviços que um sistema deve fornecer e também nas restrições desses serviços.

Vimos que, em um projeto, todos que tenham algum envolvimento nos requisitos são chamados interessados, ou *stakeholders*, e sua colaboração é fundamental para o entendimento das necessidades do negócio.

O completo entendimento das necessidades e do domínio do problema é fator de sucesso em um projeto de desenvolvimento de um sistema de *software*.

A Engenharia de Requisitos nos provém um conjunto de métodos, procedimentos e ferramentas para descobrir, analisar, documentar, verificar e validar os requisitos de um sistema (SOMMERVILLE, 2010). A Engenharia de Requisitos é um ramo da Engenharia de Software que envolve, dentro do ciclo de vida de um *software*, atividades relacionadas a requisitos (KOTONYA; SOMMERVILLE, 1998).

Os requisitos são classificados em dois grupos: requisitos de usuário e requisitos de sistema. Essa classificação leva em consideração a necessidade de expressar requisitos para diversas pessoas, com diversos pontos de vista.

Existem ainda dois tipos de requisitos: requisitos funcionais, que descrevem o comportamento do sistema, suas restrições e resultados esperados, e requisitos não funcionais, também chamados de atributos de qualidade, que descrevem as restrições nos serviços prestados pelo sistema.

O processo de engenharia de requisitos tem como objetivo obter requisitos definidos, especificados e modelos de sistema a partir de fontes de requisitos e é composto basicamente por cinco atividades: elicitação, análise, documentação e validação, sendo a atividade de gerenciamento uma atividade que apoia todo o processo.

Vimos alguns métodos para elicitação de requisitos, como entrevistas, observação do ambiente do usuário e reuniões dirigidas, e debatemos a modelagem de caso de uso como um método para análise dos requisitos elicitados e produção de um modelo conceitual.

O modelo conceitual, ou de entendimento do domínio, é uma representação do escopo do projeto, utilizado ao longo de todo ciclo de vida do *software* e por todos os envolvidos no projeto: usuários, analistas, arquitetos e gerentes. Esta é a razão pela qual temos a preocupação em definir um modelo o mais completo possível para descrição dos casos de uso e da definição de uma linguagem padrão para representação.

O modelo de descrição de caso de uso proposto do Cockburn (2005) e o diagrama de caso de uso da UML são importantes ferramentas de apoio na atividade de desenvolvimento do modelo conceitual.

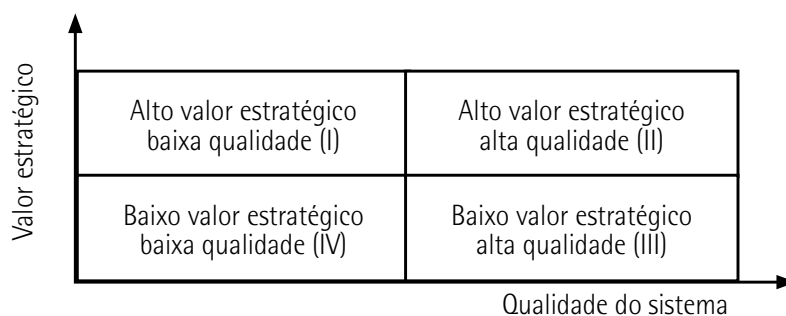
É comum que os requisitos elicitados, analisados e documentados mudem no decorrer do projeto, seja por erro, mudança da organização ou mudanças legais. A gestão de requisitos é uma atividade que tem por objetivo auxiliar na mudança desses requisitos, apontando e mitigando os riscos dessas mudanças.

Vimos ainda a matriz de rastreabilidade como uma ferramenta importante no apoio da tarefa de identificar possíveis dependências entre os requisitos alterados e outros requisitos ou artefatos.



Exercícios

Questão 1. (Enade 2008) Um ponto crítico para as organizações é a gerência de seus sistemas legados. Quanto a esses sistemas, é importante decidir se eles devem sofrer uma reengenharia, sendo reimplementados, ou não. Essa decisão é tomada após se avaliarem os sistemas legados com base em dois parâmetros: valor estratégico para a organização, ou seja, o valor que ele agrega para os serviços e produtos da organização; e qualidade do sistema, ou seja, o custo de manutenção, uma vez que sistemas de baixa qualidade possuem alto custo de manutenção. Essa avaliação classifica esses sistemas de acordo com as situações de I a IV indicadas a seguir.



Em qual(ais) dessas situações um sistema legado deve ser classificado para ser indicado a uma reengenharia?

- A) Apenas na situação I.
- B) Apenas na situação IV.
- C) Apenas nas situações I e II.
- D) Apenas nas situações II e III.
- E) Apenas nas situações III e IV.

Resposta correta: alternativa A.

Análise da questão

Sistemas legados são definidos como sistemas de importância estratégica para a empresa, e que estão em funcionamento há muito tempo – tipicamente, mais de dez anos. Esses sistemas muitas vezes estão tecnologicamente ultrapassados, além de terem sofrido muitas manutenções ao longo do tempo. A reengenharia de um sistema desse tipo (e sua reimplementação) é um risco – o novo sistema pode não atender às funcionalidades necessárias devido à ausência de documentação dos requisitos e/ou de alterações sofridas ao longo do tempo. Ou ainda porque regras corporativas podem estar registradas apenas no *software*, e novamente sua ausência nos documentos manipulados para o novo sistema implicará num sistema que não atende completamente às expectativas.

Do ponto de vista puramente técnico, apenas sistemas de baixa qualidade seriam candidatos à reengenharia, pois o próprio texto da questão explicita que "sistemas de baixa qualidade possuem alto custo de manutenção" – e uma reengenharia de tais sistemas levaria à diminuição de custos com manutenção.

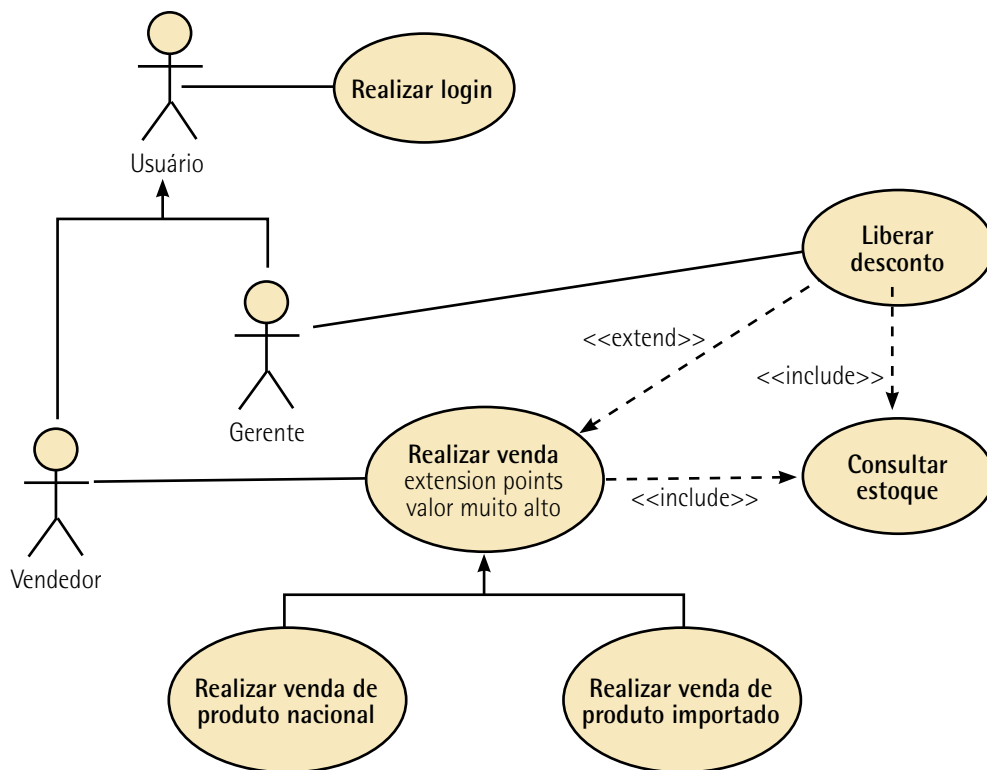
Podemos deduzir que sistemas de alta qualidade possuem baixo custo de manutenção, então não haveria indicação de reengenharia/reimplementação para esses sistemas.

Do ponto de vista estratégico, sistemas considerados de alto valor estratégico indicam uma dependência da empresa em relação a eles – portanto, devem continuar ativos. Sistemas de baixo valor estratégico têm pouca importância para os negócios da empresa.

Combinando as avaliações dos pontos de vista técnico e estratégico, podemos então afirmar que sistemas com baixa qualidade técnica e alto valor estratégico são candidatos à reengenharia e reimplementação. Portanto, a resposta correta é a alternativa A.

Fonte: SOMMERVILLE, I. *Engenharia de Software*. 8. ed. Addison-Wesley, 2007.

Questão 2. (Enade 2011) No desenvolvimento de um *software* para um sistema de venda de produtos nacionais e importados, o analista gerou o diagrama de casos de uso a seguir:



Da análise do diagrama, conclui-se que

- A) A execução do caso de uso "Consultar estoque" incorpora opcionalmente o caso de uso "Liberar desconto".
- B) A execução do caso de uso "Liberar desconto" incorpora opcionalmente o caso de uso "Realizar venda".
- C) A execução do caso de uso "Realizar venda" incorpora obrigatoriamente o caso de uso "Consultar estoque".
- D) A execução do caso de uso "Realizar venda de produto nacional" incorpora obrigatoriamente o caso de uso "Liberar desconto".
- E) Um gerente pode interagir com o caso de uso "Realizar venda", pois ele é um usuário.

Resolução desta questão na plataforma.