# Gerência de Requisitos de Software

Prof. Diego Martins Polla de Moraes
Prof. Luís Guerreiro Lopes

**Indaial – 2023**
1ª Edição

*Elaboração:*
*Prof. Diego Martins Polla de Moraes*
*Prof. Luís Guerreiro Lopes*



*Revisão, Diagramação e Produção:*
*Equipe Desenvolvimento de Conteúdos EdTech*
*Centro Universitário Leonardo da Vinci – UNIASSELVI*

*Ficha catalográfica elaborada pela equipe Conteúdos EdTech UNIASSELVI*

**C397 CENTRO UNIVERSITÁRIO LEONARDO DA VINCI.**
Núcleo de Educação a Distância. **MORAES**, Diego Martins Polla de.

**Gerência de Requisitos de Software**. Diego Martins Polla de Moraes; Luís Guerreiro Lopes. Indaial - SC: Arqué, 2023.

**210p**.

ISBN 978-65-5646-573-9
ISBN Digital 978-65-5646-569-2

"Graduação - EaD".
1. Gerência 2. Requisitos 3. Software

CDD 005.12

Bibliotecário: João Vivaldo de Souza CRB- 9-1679

Impresso por:

# APRESENTAÇÃO

O desenvolvimento de software é uma atividade complexa que envolve uma série de etapas, integração de pessoas, de negócios e de tecnologias. O desafio de construir um software de qualidade, que resolve os problemas do negócio do cliente e que traz diferenciais que permitam que este negócio continue crescendo, é almejado por todas as organizações.

Os acadêmicos de engenharia de software, na maioria das vezes, sentem-se mais entusiasmados pelos estudos de algoritmos, linguagens de programação modernas e desenvolvimento *web* e *mobile*. No entanto, dentro da engenharia de software, uma das disciplinas mais importantes é a gerência de requisitos. Por meio dela é que serão definidas as estruturas dos sistemas que serão construídos. De nada adianta construir um código fonte, uma interface agradável para o usuário, se não é aquilo que ele precisa para resolver suas atividades do dia a dia.

Para abordarmos este conteúdo de forma completa, este livro foi estruturado em 3 unidades: na Unidade 1, abordaremos os conceitos introdutórios em relação à engenharia de requisitos, seus fundamentos, a importância da engenharia de requisitos no contexto do desenvolvimento de software, as causas de falhas e riscos em requisitos de software, os processos de engenharia de requisitos, de gestão das partes interessadas, de gerenciamento de requisitos e os métodos para elicitação de requisitos, utilizando BPMN, *Design Thinking* e outras técnicas de levantamento de requisitos.

Em seguida, na Unidade 2, estudaremos como os requisitos podem ser classificados, as formas de se preparar uma documentação de requisitos, construir sua especificação, as histórias de usuário e os diagramas da UML – *Unified Model Language*.

Por fim, na Unidade 3, aprenderemos os conceitos de qualidade de requisitos, com a validação e verificação, do gerenciamento de requisitos, sobre o ciclo de vida deles, da rastreabilidade e como funcionam as mudanças em requisitos. Para finalizar a unidade, veremos sobre o mercado de trabalho em engenharia de requisitos e as ferramentas que podem ser utilizadas pelo engenheiro.

Bons estudos!

Prof. Diego Martins Polla de Moraes

# GIO

**Olá, eu sou a Gio!**

No livro didático, você encontrará blocos com informações adicionais – muitas vezes essenciais para o seu entendimento acadêmico como um todo. Eu ajudarei você a entender melhor o que são essas informações adicionais e por que você poderá se beneficiar ao fazer a leitura dessas informações durante o estudo do livro. Ela trará informações adicionais e outras fontes de conhecimento que complementam o assunto estudado em questão.

Na Educação a Distância, o livro impresso, entregue a todos os acadêmicos desde 2005, é o material-base da disciplina. A partir de 2021, além de nossos livros estarem com um novo visual – com um formato mais prático, que cabe na bolsa e facilita a leitura –, prepare-se para uma jornada também digital, em que você pode acompanhar os recursos adicionais disponibilizados através dos QR Codes ao longo deste livro. O conteúdo continua na íntegra, mas a estrutura interna foi aperfeiçoada com uma nova diagramação no texto, aproveitando ao máximo o espaço da página – o que também contribui para diminuir a extração de árvores para produção de folhas de papel, por exemplo.

Preocupados com o impacto de ações sobre o meio ambiente, apresentamos também este livro no formato digital. Portanto, acadêmico, agora você tem a possibilidade de estudar com versatilidade nas telas do celular, tablet ou computador.

Preparamos também um novo layout. Diante disso, você verá frequentemente o novo visual adquirido. Todos esses ajustes foram pensados a partir de relatos que recebemos nas pesquisas institucionais sobre os materiais impressos, para que você, nossa maior prioridade, possa continuar os seus estudos com um material atualizado e de qualidade.

# QR CODE

Olá, acadêmico! Para melhorar a qualidade dos materiais ofertados a você – e dinamizar, ainda mais, os seus estudos –, nós disponibilizamos uma diversidade de QR Codes completamente gratuitos e que nunca expiram. O QR Code é um código que permite que você acesse um conteúdo interativo relacionado ao tema que você está estudando. Para utilizar essa ferramenta, acesse as lojas de aplicativos e baixe um leitor de QR Code. Depois, é só aproveitar essa facilidade para aprimorar os seus estudos.

# ENADE

Acadêmico, você sabe o que é o ENADE? O Enade é um dos meios avaliativos dos cursos superiores no sistema federal de educação superior. Todos os estudantes estão habilitados a participar do ENADE (ingressantes e concluintes das áreas e cursos a serem avaliados). Diante disso, preparamos um conteúdo simples e objetivo para complementar a sua compreensão acerca do ENADE. Confira, acessando o QR Code a seguir. Boa leitura!

# LEMBRETE

Olá, acadêmico! Iniciamos agora mais uma disciplina e com ela um novo conhecimento.

Com o objetivo de enriquecer seu conhecimento, construímos, além do livro que está em suas mãos, uma rica trilha de aprendizagem, por meio dela você terá contato com o vídeo da disciplina, o objeto de aprendizagem, materiais complementares, entre outros, todos pensados e construídos na intenção de auxiliar seu crescimento.

Acesse o QR Code, que levará ao AVA, e veja as novidades que preparamos para seu estudo.

Conte conosco, estaremos juntos nesta caminhada!

# SUMÁRIO

# FUNDAMENTOS E TÉCNICAS DE LEVANTAMENTO DE REQUISITOS

## OBJETIVOS DE APRENDIZAGEM

**A partir do estudo desta unidade, você deverá ser capaz de**:

- compreender os conceitos de engenharia de requisitos e seus processos;
- reconhecer a relevância da engenharia de requisitos para o processo de desenvolvimento de software como um todo.
- analisar a importância que a gestão das partes interessadas tem sobre o processo de engenharia de requisitos e o desenvolvimento do software;
- aprender as técnicas de elicitação de requisitos, sabendo qual técnica ou qual combinação de técnicas utilizar em cada contexto de projeto;

## PLANO DE ESTUDOS

A cada tópico desta unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TEMA DE APRENDIZAGEM 1 – FUNDAMENTOS DA ENGENHARIA DE REQUISITOS

TEMA DE APRENDIZAGEM 2 – PROCESSOS DE ENGENHARIA DE REQUISITOS

TEMA DE APRENDIZAGEM 3 – ELICITAÇÃO DE REQUISITOS

## CHAMADA

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

# CONFIRA A TRILHA DA UNIDADE 1!

Acesse o
QR Code abaixo:

# FUNDAMENTOS DA ENGENHARIA DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 1, abordaremos os conceitos iniciais, os fundamentos relacionados à engenharia de requisitos, para entender como estas atividades permitem que o desenvolvimento de software ocorra e atenda os objetivos que as organizações almejam. A atividade de engenharia de requisitos é complexa, e envolve capacidades técnicas e habilidades interpessoais dos analistas que executam estas atividades.

Na sequência, o conceito de requisitos é apresentado e enfatizamos a relevância que a engenharia de requisitos tem no processo de desenvolvimento de software. Pontuamos os modelos de processo de desenvolvimento de software mais comuns e como as atividades de requisitos se enquadram neles, analisando suas vantagens e restrições.

Por fim, estudaremos as causas de falhas em requisitos de software, as mais comuns e como podemos tomar medidas para não ocorram nos projetos de desenvolvimento.

## 2 FUNDAMENTOS DA ENGENHARIA DE REQUISITOS

Em um curso de engenharia de software, os alunos são preparados para serem, além de engenheiros de software, analistas de sistemas, ofício que engloba uma série de atividades do ciclo completo de desenvolvimento de software. Muitas vezes, os alunos procuram um curso de engenharia de software muito focados em se tornar desenvolvedores de software, programadores ou *devs*, como popularmente são chamados atualmente. No entanto, existe um caminho para um software ser desenvolvido que não inicia na programação.

Pressman e Maxim (2016) alertam que muitos desenvolvedores de software, na sua empolgação de construí-los imediatamente, podem iniciar o desenvolvimento com informações básicas e, à medida que as etapas são entregues, as partes interessadas terão mais clareza para entender as necessidades do software. Ocorre que esta prática fará com que todo o ciclo do software seja executado, sem que se tenha uma certeza das necessidades que aquele software precisará atender.

Para que uma demanda possa estar apta a ser construída por um desenvolvedor de software e para esteja madura para ser desenvolvida, uma série de passos precisa ser executados. Uma necessidade deve ser entendida, trabalhada, detalhada por um profissional de engenharia de requisitos para que se possa entender essas necessidades e convertê-las em uma documentação que possa ser trabalhada por uma equipe de desenvolvimento. A informatização, a sistematização, a transformação digital não ocorre com iniciativas apenas do pessoal de tecnologia. No mundo dos negócios é que estão as necessidades de se trabalhar a informatização e a construção de sistemas que atendam estas necessidades.

Além das necessidades das organizações, atualmente, temos as necessidades da população geral. As pessoas, com seus celulares e outros dispositivos conectados, resolvem suas questões do dia a dia, como se comunicar, pagar contas, comprar bens e serviços. Pressman e Maxim (2016) afirmam que a computação onipresente permite que a tecnologia avance e esteja integrada a muitos objetos do cotidiano, permitindo que eles sejam ligados em rede.

A computação onipresente tem o potencial de melhorar a qualidade de vida das pessoas em diversas áreas, como automação residencial, saúde e bem-estar. Com dispositivos conectados, é possível, por exemplo, monitorar o estado de saúde de pacientes à distância e fornecer um atendimento mais personalizado. A tecnologia também pode ser utilizada para melhorar a mobilidade urbana, tornando o transporte mais seguro e eficiente.

A definição de requisitos é um processo crítico e fundamental também para o desenvolvimento de sistemas de computação onipresentes. Esses sistemas envolvem uma grande variedade de dispositivos, softwares e redes de comunicação, e é preciso garantir que todos esses componentes estejam integrados de forma adequada para fornecer uma experiência de uso coesa e intuitiva para os usuários.

Os requisitos para computação onipresente podem variar de acordo com as necessidades específicas dos usuários e das aplicações. Por exemplo, em sistemas de automação residencial, os requisitos podem incluir a capacidade de controlar remotamente os dispositivos domésticos, como luzes e eletrodomésticos, além de integrar sensores para monitorar o ambiente e garantir o conforto e segurança dos moradores.

Já em sistemas de saúde, os requisitos podem incluir a capacidade de monitorar a condição física dos pacientes e disparar alertas em tempo real para médicos e cuidadores em caso de emergência. É importante que os requisitos sejam claramente definidos e documentados desde o início do projeto, para garantir que o sistema seja desenvolvido de forma eficiente e atenda às expectativas dos usuários.

Para definir os requisitos para computação onipresente, é necessário um entendimento profundo dos usuários finais, suas necessidades e desafios, além das tecnologias e recursos disponíveis. Isso pode ser realizado através de pesquisas de mercado, entrevistas com usuários, prototipação e testes de usabilidade. O objetivo é criar uma lista clara e completa de requisitos que possam orientar o desenvolvimento do sistema e garantir que ele atenda às necessidades dos usuários de forma eficiente e satisfatória.

Sistemas organizacionais e sistemas de computação onipresente são importantes para as empresas e para toda a sociedade. Enquanto os primeiros gerenciam processos internos, os segundos têm o potencial de mudar a forma como as pessoas interagem com a tecnologia e o ambiente que as rodeiam. Ambos são complementares na busca por soluções eficientes e inovadoras. Empresas que adotam e integram essas tecnologias em suas operações e estratégias têm mais chances de se destacarem no mercado e oferecerem uma experiência diferenciada para seus clientes e usuários. Para isso ocorra, as definições de requisitos são primordiais.

É importante lembrar que todo sistema tem um ciclo de vida. Ele nasce neste processo de engenharia de requisitos, é construído, inicia-se seu uso, e depois ele tem uma evolução, manutenção. Um sistema pode ficar em uso por três, cinco, dez, e, em alguns casos, até mais de vinte anos. Os requisitos são fontes importantes para que sistema possa continuar em uso sem perder suas capacidades, propriedades, regras e objetivos.

O ciclo de vida de desenvolvimento de software é um processo contínuo que envolve diferentes fases, desde a concepção da ideia até a entrega do produto final. De acordo com Sommervile (2018, p. 31):

> O ciclo de vida de desenvolvimento de software é uma representação simplificada de um processo de software. Cada modelo representa um processo a partir de uma perspectiva particular e, desse modo, fornece apenas informações parciais sobre esse processo. [...] Esses modelos genéricos são descrições mais gerais e abstratas dos processos de software, e podem ser utilizados para explicar as diferentes abordagens ao desenvolvimento de software.

As fases do ciclo de vida podem variar de acordo com o modelo adotado pela empresa ou equipe de desenvolvimento, mas geralmente incluem atividades como análise de requisitos, design, codificação, testes e manutenção. É importante ressaltar que o processo de desenvolvimento de software é iterativo e pode ser adaptado de acordo com as necessidades e feedbacks do cliente e dos usuários.

O objetivo do ciclo de vida de desenvolvimento de software é garantir que o produto final atenda às expectativas e necessidades dos usuários, além de ser entregue dentro do prazo e do orçamento estabelecidos (PMI, 2017). Um processo de desenvolvimento de software adequado é capaz de alinhar as necessidades do cliente com a tecnologia disponível e com as habilidades da equipe de desenvolvimento.

Dentro dos processos que compõe o ciclo de desenvolvimento de software, a Engenharia de Requisitos é uma das fases mais importantes. É imperativo ponderar que todas as etapas futuras da construção de um software se basearão nos requisitos definidos nesta fase. Erros nos processos de requisitos resultarão em falhas sucessivas em todas as fases posteriores do projeto de software (KERR, 2015).

Para Pressman e Maxim (2016), a engenharia de requisitos constitui uma base concreta para os projetos de software. O software que não tem um processo de engenharia de requisitos adequado tem grande possibilidade de não atender as necessidades do cliente.

Um conceito completo de engenharia de requisitos é apresentado por Vazquez e Simões (2016, p.17): "uma disciplina da engenharia de software que consiste no uso sistemático e repetitivo de técnicas para cobrir atividades de obtenção, documentação e manutenção de um conjunto de requisitos para software que atendem os objetivos de negócio e sejam de qualidade".

Um ponto de vista complementar é apresentado por Softex (2021), incluindo que as inconsistências entre os requisitos, os planos e os produtos de trabalho precisam ser identificados e tratados no contexto da engenharia de requisitos.

É comum que surjam inconsistências entre os requisitos, os planos e os produtos de trabalho durante o processo de desenvolvimento de software. Essas inconsistências podem causar atrasos, retrabalho e, em última instância, afetar a qualidade do produto final. Por isso, é fundamental que a engenharia de requisitos seja aplicada de forma sistemática e rigorosa durante todo o ciclo de vida de desenvolvimento de software, desde a concepção da ideia até a entrega do produto.

Identificar e tratar as inconsistências entre os requisitos, os planos e os produtos de trabalho no contexto da engenharia de requisitos é fundamental para garantir que o produto final atenda às expectativas e necessidades dos usuários, além de ser entregue dentro do prazo e do orçamento estabelecidos.

Um exemplo comum de inconsistência de requisitos é quando um requisito não foi claramente definido ou especificado de forma inadequada, resultando em diferentes interpretações por parte dos desenvolvedores e usuários-chave. Por exemplo, imagine que um requisito para um sistema de gerenciamento de estoque é "o sistema deve ser capaz de lidar com grandes volumes de transações". Essa declaração pode ser interpretada de diferentes formas pelos desenvolvedores, podendo gerar soluções muito diferentes. Para alguns, "grandes volumes" pode significar um grande número de transações diárias, enquanto para outros pode significar um grande número de transações por segundo. Qual a definição exata para “grandes volumes”? Para um pode ser 1.000, para outros 1.000.000, para outros 100.000.000 de transações, fica indefinido. A interpretação de um requisito deve ser consistente independentemente de quem o leia.

Uma grande questão envolvida é que os engenheiros de requisitos não conhecem os problemas e capacidades que os sistemas precisam abranger. Afinal, estes profissionais são especialistas em coletar, especificar, validar e implantar sistemas que atendem as necessidades das organizações. Um engenheiro de requisitos não tem a obrigação de conhecer detalhadamente como as organizações e seus negócios funcionam para que possa projetar os sistemas que os atendem. É elementar que o conhecimento de noções de gestão, de processos e de ambiente organizacional é requerido, mas as organizações é que precisam ter conhecimento de suas necessidades de forma clara para que o analista de requisitos faça o seu trabalho de forma adequada (POHL; RUPP, 2011).

Para que isso ocorra, é necessário que as informações sejam fornecidas pelas partes interessadas, que são os envolvidos no processo que aquele software pretende automatizar. Partes interessadas são pessoas ou organizações que tem influência (direta ou indireta) nos requisitos de um sistema (POHL; RUPP, 2011)

É importante ressaltar que a maioria dos fornecedores de requisitos nunca participou de um projeto de desenvolvimento de um software. Eles são especialistas na área que atuam, são advogados, administradores, contadores, professores, médicos, profissionais das mais diversas áreas de atuação que irão ser envolvidos numa fase essencial do desenvolvimento de um projeto de software, trazendo as necessidades do mundo real das quais se pretende resolver através do projeto de um software (SOMMERVILLE, 2018).

O engajamento das partes interessadas no processo de desenvolvimento de software é crucial para garantir que o produto final atenda às expectativas e necessidades dos usuários. De acordo com o PMI (2017), o engajamento das partes interessadas é uma das áreas de conhecimento chave da gestão de projetos e envolve a identificação, planejamento, gerenciamento e comunicação com as partes interessadas ao longo de todo o ciclo de vida do projeto.

A falta de engajamento das partes interessadas pode levar a requisitos incompletos, mudanças de escopo inesperadas e insatisfação dos usuários finais. Por outro lado, um alto nível de engajamento das partes interessadas pode levar a uma melhor compreensão das necessidades dos usuários, uma maior colaboração entre as partes interessadas e uma maior probabilidade de sucesso do projeto.

Para engajar efetivamente as partes interessadas, é importante estabelecer uma comunicação clara e eficaz, fornecer informações relevantes e oportunas sobre o projeto e suas implicações e considerar as opiniões e perspectivas das partes interessadas durante todo o processo de desenvolvimento. O uso de ferramentas e técnicas de comunicação adequadas, como reuniões regulares, pesquisas de satisfação e questionários de feedback, pode ajudar a garantir que as partes interessadas estejam envolvidas e informadas ao longo de todo o processo.

Um exemplo de falta de engajamento de partes interessadas em um projeto de software pode ocorrer quando um cliente solicita um novo recurso ou melhoria para um software, mas não fornece feedbacks adequados sobre sua utilização ou não participa ativamente dos testes do software durante o processo de desenvolvimento.

É importante convencionar que os requisitos não são de uma única pessoa ou área da organização. Eles tipicamente são uma combinação entrelaçada de necessidades de várias pessoas e de diferentes níveis organizacionais que estão vinculados a um processo de trabalho ou a um conjunto de processos de trabalho. Um outro fator preponderante são as normas organizacionais e requisitos legais que podem estar vinculados a estas necessidades (POHL; RUPP, 2011).

Processo de trabalho refere-se a uma série de etapas e atividades necessárias para realizar uma tarefa específica ou produzir um produto ou serviço. Ele envolve a aplicação de conhecimentos, habilidades e recursos para alcançar um objetivo definido. Um processo de trabalho pode variar em complexidade e pode ser aplicado em diversas áreas, como produção industrial, serviços, educação, pesquisa, saúde, entre outras.

A eficiência do processo de trabalho é um fator importante para o sucesso de uma empresa ou organização, uma vez que pode afetar diretamente a qualidade do produto ou serviço, o prazo de entrega, o custo e a satisfação do cliente. Por isso, muitas empresas buscam constantemente aprimorar seus processos de trabalho através da construção de sistemas computacionais que permitam executá-los da melhor forma.

De acordo com Pohl e Rupp (2011), as normas organizacionais são fundamentais para garantir que as atividades de uma empresa sejam conduzidas de maneira consistente e eficiente. Essas normas geralmente são estabelecidas pelo alto escalão da organização e podem incluir políticas, procedimentos e padrões que devem ser seguidos por todos os funcionários e por consequência os sistemas que aquela organização

utiliza. Além disso, as normas organizacionais também podem ser influenciadas por fatores externos, como as leis e regulamentações que regem o mercado que aquela organização opera.

No que diz respeito aos requisitos legais, é importante que as empresas estejam cientes das leis e regulamentações aplicáveis para conduzir os seus negócios de maneira eficaz. Conforme apontado por Biesdorf, Court e Willmott (2013), esses requisitos podem variar de acordo com o mercado e a localização dos negócios, mas geralmente incluem leis relacionadas a questões tributárias, trabalhistas, ambientais e de segurança. É crucial que os sistemas construídos estejam em conformidade com todas as leis e regulamentações aplicáveis. O não cumprimento desses requisitos pode resultar em multas, sanções ou outras consequências negativas para a empresa.

**DICAS**

Se as partes interessadas não têm ou têm pouca experiência com projetos de desenvolvimento de software, é interessante promover um Workshop com o grupo, demonstrando como o processo de desenvolvimento de software funciona, e qual a importância das informações e o nível de engajamento de cada um.

Esta prática permitirá que a Engenharia de Requisitos seja minimamente entendida pelos envolvidos e a chance de sucesso no seu projeto aumentará consideravelmente. Este movimento também contribuirá com seu envolvimento com as partes interessadas. Provavelmente confiarão mais no trabalho da equipe de software após estes contatos iniciais.

## 2.1 CONCEITO DE REQUISITOS

Sommervile (2018) afirma que os requisitos de um sistema são as definições dos serviços que este sistema deve prestar a seus usuários e também as restrições a sua operação. Isso significa que os requisitos representam o que os usuários esperam daquele sistema.

Complementando este conceito, o IEEE (2014) apresenta que os sistemas precisam atender necessidades e resolver problemas que estão presentes no mundo real, permitindo que seus usuários atinjam seus objetivos.

Para Sommerville (2018), um ponto muito importante é escrever os requisitos em diferentes níveis de refinamento, diferentes linguagens. Isso abrange partes interessadas com níveis de conhecimento diferentes em relação ao negócio, e níveis diferentes em relação ao sistema, facilitando assim o entendimento de todos em relação ao sistema a ser construído.

A abordagem de escrever os requisitos em diferentes níveis de detalhamento é conhecida como especificação progressiva e pode ser utilizada para evitar erros e ambiguidades na definição dos requisitos de software.

Um exemplo de especificação progressiva pode ser aplicado na definição dos requisitos para um sistema de vendas online. Inicialmente, pode-se definir um requisito geral, como "o sistema deve permitir a venda de produtos pela internet". Em seguida, pode-se especificar mais detalhadamente os requisitos funcionais, como "o sistema deve permitir a pesquisa de produtos por categoria e palavra-chave, o cadastro de clientes, a inclusão de produtos no carrinho de compras, a realização de pagamentos via cartão de crédito e boleto bancário, além de gerar um comprovante de compra para o cliente".

Com a aplicação da especificação progressiva, é possível garantir que os requisitos do sistema de vendas online estejam completos e corretos, além de permitir que diferentes partes interessadas, como desenvolvedores, gerentes de projeto e usuários finais, compreendam os requisitos de forma clara e precisa.

Os requisitos são ponto essencial para definição do escopo de um sistema. Pode parecer simples, que o escopo de um sistema seja definido a partir do contexto que precisa ser automatizado/informatizado. Por exemplo, o escopo é construir um sistema de controle de um restaurante delivery. Existem várias formas de se trabalhar um restaurante delivery, com diferentes volumes de demanda, variedade de cardápio, formas de entrega. Também existem diferentes níveis de maturidade das empresas, o que muda as necessidades de gerenciamento através de um sistema.

Os processos de engenharia de requisitos permitirão que os requisitos sejam conhecidos, detalhados, especificados, validados e posteriormente implementados no sistema em questão (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2018):

Conhecidos: é preciso identificar e entender as necessidades e expectativas dos usuários do sistema, bem como as restrições e limitações a serem consideradas. Isso pode ser feito por meio de entrevistas, questionários, reuniões e outras técnicas de elicitação de requisitos.

Detalhados: os requisitos devem ser detalhados o suficiente para que possam ser compreendidos pelas partes interessadas, usuários, desenvolvedores, analistas e testadores que trabalharão no projeto. É importante que os requisitos estejam completos e bem definidos, sem ambiguidades ou contradições.

Especificados: após a identificação e detalhamento dos requisitos, é preciso escrevê-los de forma clara e concisa, utilizando uma linguagem adequada e padronizada. Isso pode ser feito por meio de documentos, diagramas e outros artefatos de especificação de requisitos.

Validados: a validação dos requisitos tem como objetivo garantir que eles atendam às necessidades e expectativas dos usuários, bem como às restrições e limitações do projeto. Isso pode ser feito por meio de revisões, validações formais e testes de aceitação.

Implementados: após a validação dos requisitos, é possível iniciar a implementação do sistema, que deverá ser feita de acordo com as especificações e padrões definidos na fase anterior.

## 2.2 A RELEVÂNCIA DA ENGENHARIA DE REQUISITOS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

A engenharia de requisitos é fase essencial para construção de um software. Todas as fases que a sucedem se baseiam no resultado do trabalho dos processos de engenharia de requisitos. Para Drucker (1963), certamente, não há nada tão inútil quanto fazer com grande eficiência o que não deve ser feito.

Suponha que uma empresa esteja desenvolvendo um software de gestão financeira para pequenas empresas e inicie o projeto sem realizar uma fase adequada de engenharia de requisitos. Sem um processo claro de definição de requisitos, a equipe de desenvolvimento pode acabar criando funcionalidades que não são necessárias ou importantes para o público-alvo do software, enquanto negligência recursos e funções essenciais.

Por exemplo, eles podem gastar muito tempo e esforço para desenvolver um módulo avançado de relatórios financeiros, enquanto negligenciam recursos importantes, como a integração com serviços bancários on-line ou a emissão de recibos fiscais. Nesse caso, eles estariam trabalhando com grande eficiência, mas no final das contas, estariam construindo um software que não atende às necessidades reais dos usuários e que pode até mesmo falhar em satisfazer as expectativas do mercado.

É inútil construir um software sem antes entender e definir claramente o que deve ser feito. A engenharia de requisitos é a fase inicial e essencial para garantir que o software atenda às necessidades dos usuários e do mercado-alvo, evitando que a equipe de desenvolvimento gaste tempo e recursos em funcionalidades desnecessárias ou irrelevantes. Negligenciar as atividades de engenharia de requisitos em um projeto de software traz como consequências atrasos de cronograma, aumento nos custos, alto nível de defeitos no software “pronto” e o que é considerado pior: a entrega de um software que não atende plenamente as necessidades de suas partes interessadas (VAZQUEZ; SIMÕES, 2016).

A engenharia de requisitos também contribui para a qualidade do software. Ao definir requisitos claros e precisos, é possível evitar ambiguidades e inconsistências que podem levar a erros e defeitos. Isso também facilita a validação e verificação dos requisitos, permitindo que sejam identificados e corrigidos problemas antes da implementação (SOMMERVILLE, 2018).

A qualidade do software está intimamente ligada à engenharia de requisitos, como afirmaram Wiegers e Beatty (2013). Isso significa que, para garantir um produto de software de alta qualidade, é crucial estabelecer requisitos claros, precisos e bem definidos desde o início do processo de desenvolvimento. Além disso, é importante que esses requisitos sejam validados e verificados continuamente durante todo o ciclo de vida do projeto, a fim de garantir que eles atendam às necessidades do usuário e estejam alinhados com os objetivos do negócio.

Quando falamos de defeitos, é importante ressaltar que quanto mais tarde um defeito em um software é identificado, mais caro o esforço para corrigir aquele defeito ficará. Se aquele software já estiver em produção, os custos podem envolver além do trabalho de desenvolvimento de software, prejuízos em relação ao negócio (VASZQUEZ; SIMÕES, 2016).

Estudos apresentados por Bohem (1981 *apud* KERR, 2015, p. 11) apresentam que:

> Quanto mais tempo um erro nos requisitos demorar para ser identificado e corrigido, maiores serão seus custos. O trabalho necessário para corrigir um erro na programação chega a ser 20 vezes maior do que o necessário na engenharia de requisitos, e se ele só se tornar visível durante os testes de aceitação, o trabalho pode chegar a ser até cem vezes maior.

Por exemplo, se durante o levantamento de requisitos de um software de vendas de produtos no varejo, o contador da empresa repassou as regras de cálculo de impostos, considerando um percentual de 17%, toda a formação de preços e cálculos de impostos foi realizado considerando tal requisito. Após 3 meses de operação do sistema, em uma auditoria, verificou-se uma discrepância de valores, e que a origem estava no recolhimento de impostos, que deveria ser de 22% e não de 17%. Para um faturamento de R$ 1.000.000,00 (um milhão de reais), a diferença entre dos cálculos chega a R$ 50.000,00 (cinquenta mil reais), além do custo para ajustar o software considerando a alíquota correta. Caso este problema tivesse sido descoberto em fases de validação de requisitos, bastaria corrigir os documentos de especificação de requisitos que continham a regra da forma incorreta.

As falhas em softwares são comuns, como falhas em qualquer produto. As falhas na construção do software podem ser facilmente descobertas em etapas de testes do software. Se o requisito tal definido para aquele software não for atendido na sua entrega, este problema será reportado e corrigido. O problema maior é quando a

falha é nos requisitos. O sistema faz corretamente o que os requisitos exigem. A questão é se os requisitos estiverem incorretos, o software não atenderá as necessidades das partes interessadas, fazendo com que o seu objetivo não seja atingido.

## 2.3 CAUSAS DE FALHA EM REQUISITOS DE SOFTWARE

Segundo dados do PMI (2017), 47% dos projetos que tem fracasso são causados por uma falha no processo de engenharia de requisitos. Este dado abrange todos os tipos de projetos. É bem provável que se for avaliado apenas projetos de software, este percentual seja bem maior.

Uma pesquisa realizada pelo Project Management Institute (PMI, 2018) apresenta que a imprecisão dos requisitos foi apontada por 35% dos que responderam à pesquisa como uma das três causas elementares para falhas em seus projetos, ficando como a terceira mais citada. As outras causas foram mudanças nos objetivos dos projetos que foram apontadas por 37% dos respondentes, e mudanças nas prioridades organizacionais que foi apontada por 39% dos participantes.

Quando falamos sobre desenvolvimento de software, é comum conhecermos histórias de falhas nestes projetos. A maioria das falhas de projetos de software está ligada à falta ou falha nos requisitos definidos. Muitas vezes, as partes interessadas não fornecem detalhes que consideram óbvios, por fazer parte do seu dia a dia de trabalho, mas que para o Engenheiro de Requisitos são totalmente desconhecidos. Para minimizar estes problemas, a equipe de projeto deve deixar claro a importância e a função dos processos de Engenharia de Requisitos. Os *Stakeholders* devem ser envolvidos de forma plena, pois através desta inclusão mais profunda deles nos processos de requisitos, a chance de sucesso aumenta (SOMMERVILLE, 2018).

Apesar do avanço constante das ferramentas e tecnologias de engenharia de software, ainda não foi encontrada uma solução milagrosa ou "bala de prata" para a complexidade inerente ao desenvolvimento de software. Em vez disso, Brooks Jr. (1986) defende a importância de compreender a natureza essencial do problema e trabalhar diligentemente para mitigar os acidentes e problemas que inevitavelmente surgem.

Em um estudo conduzido por um grupo de pesquisadores, foram elencados 17 fatores de riscos que podem causar falhas em projetos de software. A seguir, estão relacionados os fatores que são vinculados a requisitos (SCHMIDT *et al.*, 2001):

- Um clima de mudança no ambiente empresarial e organizacional que gera instabilidade no projeto.
- Ambiente corporativo instável: as pressões competitivas alteram radicalmente os requisitos do usuário, às vezes tornando todo o projeto obsoleto.
- Falha ao identificar todas as partes interessadas: uma centralização em pessoas

físicas pode gerar uma visão estreita em relação aos requisitos.

- Não gerenciar as mudanças adequadamente: os projetos necessitam de um processo para gerenciar a mudança de modo que o escopo e o orçamento sejam controlados.
- Desvio de escopo: não se define completamente o escopo do novo sistema e os requisitos antes de começar, consequentemente não entende o verdadeiro esforço de trabalho, conjuntos de habilidades e tecnologia necessários para concluir o projeto.
- Projeto não é baseado em caso de negócios sólido: usuários e desenvolvedores ignoram os requisitos de negócios, desenvolvem o sistema em prol de alguma tecnologia que desejam aplicar.
- Falta de requisitos congelados: uma consequência da mudança nas necessidades dos usuários, são as mudanças nos requisitos. Por consequência, o sistema nunca será concluído e disponível em produção porque nenhum dos requisitos é concluído. Como alternativa, o congelamento de um subconjunto da funcionalidade e a entrega permitem a conclusão do sistema e as versões de atualização conforme necessário.
- Falta de conhecimento/habilidades necessárias no pessoal do projeto: por exemplo, tecnologia, conhecimento de negócios e experiência.

Para ficar mais claro, apresentamos a seguir alguns exemplos de falhas em requisitos que acarretarão falhas no software pronto:

- Em um software de caixa de supermercado, não há a opção de utilizar mais de um meio de pagamento, algo muito comum no dia a dia desta operação.
- Em um software de gestão educacional, quando um professor registra as presenças dos alunos, tem apenas as alternativas Presente ou Falta. Se o aluno depois apresenta um atestado médico ou qualquer outra justificativa, não há meio de registrá-lo para que a falta não seja computada. Com isso, houve alguns casos de alunos reprovados por falta no sistema, mas que na prática cumpriram o requisito mínimo de presença, descontando os dias que estiveram afastados por motivos legais.
- Em um software de e-commerce, não é possível inativar ou "despublicar" um produto. Isso faz com que produtos antigos que não tem mais estoque e não são mais comercializados há muito tempo continuem aparecendo para os clientes da loja virtual.
- Em um software de gestão qualquer, o controle de acessos determina que apenas usuários administradores possam fazer a gestão de usuários. Ocorre que um usuário Administrador excluiu todos os usuários, inclusive ele. Agora não é mais possível incluir nenhum usuário na plataforma.

Neste tema, foram abordados os conceitos e princípios essenciais para a compreensão da engenharia de requisitos. A partir da definição dos requisitos, é possível obter uma visão clara do que deve ser entregue e atender às necessidades dos usuários finais.

A compreensão da importância dos requisitos, bem como das técnicas para levantá-los, documentá-los e gerenciá-los, é fundamental para o sucesso do projeto de software. Além disso, é preciso estar atento às mudanças que possam ocorrer ao longo do ciclo de vida do software e saber como lidar com elas.

As atividades da engenharia de requisitos críticas para o sucesso de projetos de desenvolvimento de software. É fundamental que os engenheiros de requisitos entendam a importância da comunicação e colaboração com todas as partes interessadas no projeto, incluindo usuários finais, clientes e desenvolvedores. Com a abordagem correta e uma aplicação dos processos de engenharia de requisitos de acordo com o contexto específico, é possível garantir a satisfação dos usuários e o sucesso do projeto.

# RESUMO DO TÓPICO 1

**Neste tópico, você aprendeu:**

- Os fundamentos da engenharia de requisitos e como ela funciona.
- A importância que a engenharia de requisitos exerce no processo de desenvolvimento de software.
- Quais são as etapas que permeiam o processo da engenharia de requisitos.
- Negligenciar os processos de engenharia de requisitos pode causar falhas em todo o processo de desenvolvimento de software, tornando elevado os custos para correção destas falhas.

# AUTOATIVIDADE

1 A engenharia de requisitos é a fase inicial e essencial para garantir que o software atenda às necessidades dos usuários e do mercado-alvo, evitando que a equipe de desenvolvimento gaste tempo e recursos em funcionalidades desnecessárias ou irrelevantes. Com base nos estudos realizados sobre as atividades de engenharia de requisitos, com relação a sua relevância no processo de desenvolvimento de software, assinalem a alternativa CORRETA:

a) ( ) Os processos de engenharia de requisitos são opcionais, quem decide se serão executados ou não é o cliente, a depender do orçamento do projeto;
b) ( ) Um sistema pode ter seu processo de desenvolvimento iniciado pelo código-fonte, os requisitos podem ser escritos baseados nos códigos finalizados;
c) ( ) A engenharia de requisitos é fase essencial e ponto de partida para construção de qualquer software.
d) ( ) Requisitos podem ser definidos pela equipe de tecnologia, sem necessidade de envolvimento de usuários/clientes.

2 Os processos de engenharia de requisitos permitirão que estes sejam conhecidos, detalhados, especificados, validados e, posteriormente, implementados no sistema em questão. É reconhecido como essencial o envolvimento das partes interessadas nos processos de engenharia de requisitos. Com relação às partes interessadas, analise as sentenças a seguir:

I- Estão vinculadas ao processo de engenharia de requisitos pois pagarão pelos serviços executados.
II- Podem ser envolvidas apenas para validar o sistema em funcionamento, afinal não compreendem sobre tecnologia e contratam engenheiros de software para cumprir este papel.
III- Devem ser envolvidos em todo o ciclo de desenvolvimento de software, desde sua concepção, para que possam participar ativamente da definição do escopo e da validação do sistema construído.

Assinale a alternativa CORRETA:

a) ( ) As sentenças I e II estão corretas.
b) ( ) Somente a sentença II está correta.
c) ( ) As sentenças I e III estão corretas.
d) ( ) Somente a sentença III está correta.

3 As falhas de software os afetam como qualquer produto construído por um processo estruturado. Uma falha de software ocorre quando ele não tem um funcionamento adequado. Sobre o que pode ser considerado como uma falha de software, assinale a alternativa CORRETA:

a) ( ) A ausência de funcionalidades essenciais para cumprimento das atividades de usuários que fazem parte do escopo do sistema.
b) ( ) A ausência de funcionalidades que compõe as operações da organização que utiliza o sistema, mas que não estão contidas no escopo dele.
c) ( ) A ausência de facilidades contidas em softwares similares/concorrentes, mas que não interferem de modo a impossibilitar a execução das atividades.
d) ( ) O funcionamento de uma funcionalidade de acordo com as necessidades dos usuários.

4 O desenvolvimento de software é uma área muito ampla que pode abranger diferentes tipos de sistemas. Utilizando uma abordagem profissional existem diversos modelos de processos que podem ser utilizados. O que há em comum nestes modelos de processos é que todos incluem as atividades de engenharia de requisitos como primordiais para execução de suas atividades. Disserte sobre como os principais modelos de processos de desenvolvimento lidam com a engenharia de requisitos.

5 É comum existirem projetos de criação/evolução de software em que a documentação dos requisitos não existe ou é insuficiente. Neste contexto, disserte sobre os riscos que este tipo de abordagem pode causar no software produzido.

# PROCESSOS DE ENGENHARIA DE REQUISITOS

## 1 INTRODUÇÃO

No tema de Aprendizagem 2, iremos abordar os principais modelos de processo de desenvolvimento de software e como a Engenharia de Requisitos se vincula a cada um deles e quais etapas compõe a Engenharia de Requisitos. Isso permite com que os conceitos da engenharia de requisitos sejam melhor compreendidos e como eles irão se materializar num processo de desenvolvimento de software. Muitas vezes, o processo de engenharia de requisitos é abreviado em projetos de desenvolvimento de software. As consequências disso perpetuam-se por todo o processo de desenvolvimento e no produto final, o software "concluído".

Neste tema, também aprofundaremos os conhecimentos em partes interessadas, que são essenciais para a engenharia de requisitos. O desenvolvimento de software como área meio sempre irá depender das definições das partes interessadas para poder compreender as necessidades, os objetivos, usufruir dos resultados que o software proporciona.

Por fim, estudaremos os riscos em requisitos de software, considerando que eles compõem parte de um projeto e abordaremos os conceitos de gerenciamento de riscos que podem ser aplicados a área de requisitos de software.

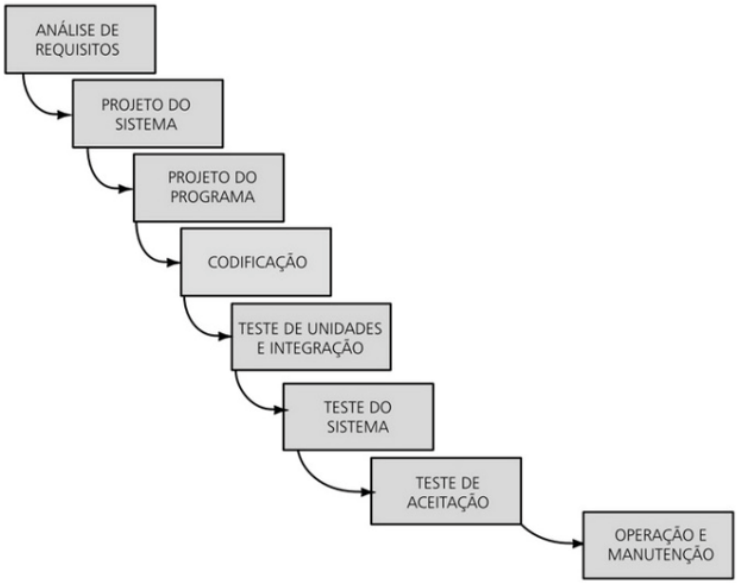
## 2 PROCESSOS DE ENGENHARIA DE REQUISITOS

Os denominados modelos pesados de processos de desenvolvimento de software, o qual podemos citar como principal o **modelo cascata**, preveem que todas as fases de processo de desenvolvimento de software ocorram em uma sequência, e apenas uma única vez (SOMMERVILLE, 2018).

Em sua obra, Pressman e Maxim (2016) afirmam que no modelo cascata, que também é chamado de **ciclo de vida clássico**, uma fase não inicia antes que a outra tenha sido concluída, aprovada e documentada. Isto faz com que a fase de requisitos seja inicial e única, ou seja, todo o processo de engenharia de requisitos ocorra de uma única vez e somente uma vez. É necessário considerar que neste cenário, todos os requisitos do software precisam ser elicitados, documentados e validados antes que a fase de projeto de software se inicie.

A Figura 1 apresenta o formato que o modelo de processo cascata propõe. O modelo não prevê opção para quando durante as fases posteriores a de requisitos sejam encontradas necessidades de modificação nos requisitos antecipadamente acordados. É considerada como a principal falha do modelo cascata - **se alinhar a um processo de fabricação e não de criação**.

Figura 1 –Modelo Cascata




Fonte: Pfleeger (2004, p. 39)

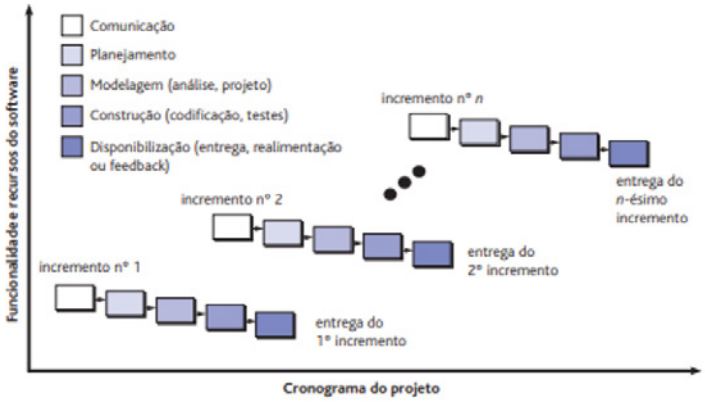
No modelo cascata não são previstos caminhos de retorno entre as atividades, o que faz com que a criação fique prejudicada. A criação de um software envolve tentativas, análise de alternativas, análise de viabilidade, aprendizado com erros para chegar a uma solução ideal para o problema (PFLEEGER, 2004)

Para Pressman e Maxim (2016), o modelo cascata pode ser aplicado com sucesso em adaptações ou aperfeiçoamentos pontuais forem ser realizados em um software, como uma adaptação originada de uma exigência legal, que possui requisitos bem definidos e relativamente estáveis. Ainda assim, o modelo tem uma tendência à falha pelos seguintes motivos:

- Projetos do mundo real raramente seguem o fluxo sequencial e rígido que o modelo cascata sugere.
- Dispor de todas as necessidades em uma fase inicial é algo impossível para a maioria das partes interessadas. Sem levar em conta a incerteza natural que faz parte da maioria dos projetos, sendo eles de software ou não.
- O cliente precisa ter paciência, afinal, a entrega de software funcional para ele ocorrerá apenas no final do projeto.

Um outro modelo mais moderno e muito utilizado atualmente é o modelo incremental. O modelo incremental é centrado na ideia de desenvolver uma implementação inicial, um fragmento do software e apresentar aos usuários, receber os feedbacks e partir para a criação de tantos outros incrementos/versões até que um software ideal esteja construído. Neste modelo, conforme pode ser verificado na Figura 2, as atividades de especificação, desenvolvimento e validação são intercaladas e iterativas (SOMMERVILLE, 2018).

Figura 2 – Modelo Incremental




Fonte: Pressman e Maxim (2016, p. 44)

Desta forma, a engenharia de requisitos é fase constante no desenvolvimento do software utilizando o modelo incremental. Definido os incrementos que o software terá, o escopo de requisitos a serem tratados naquele momento é o do incremento. Para Pressman e Maxim (2016), as vantagens do modelo incremental em relação ao modelo cascata são:

- O custo de das mudanças é reduzido. Quando há uma mudança nas definições de requisitos, a quantidade de documentação a ser refeita está limitada ao incremento.
- Considerando a baixa capacidade de abstração que as partes interessadas têm, apresentar partes do software funcionando aumenta o entendimento da solução que está sendo desenvolvida. Isso contribui para o levantamento das necessidades dos próximos incrementos.
- A entrega de partes funcionais faz com que os stakeholders possam utilizar o software mesmo sem toda funcionalidade concluída, o que não é possível quando é utilizado o modelo cascata.

Com relação aos métodos ágeis, estes utilizam essencialmente a abordagem do modelo incremental, mas com a inclusão de alguns princípios que os fazem se aproximar dos stakeholders (BECK *et al.*, 2001).

Os métodos ágeis são baseados no Manifesto Ágil, que contém 12 princípios. Os princípios que têm maior afinidade em relação a requisitos são (BECK *et al.*, 2001):

- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento.
- Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto (BECK *et al.*, 2001).

O benefício da aproximação junto às partes interessadas oferece maiores chances de se obter as informações necessárias para os requisitos do projeto, mas alerta para o fato de que muitas das metodologias ágeis serem interpretadas como um a sugestão de uma menor preocupação com documentação, o que pode tornar o registro dos requisitos precário, e dificultar a implementação dos futuros incrementos de um software. O documento de requisitos é a fonte de informações para avaliação das mudanças necessárias em um software e, portanto, ter um documento incompleto pode tornar o processo difícil e custoso (SOMMERVILLE, 2018).

Para Vazquez e Simões (2016), espalhar as atividades de requisitos ao longo de todo o desenvolvimento traz resultados melhores. Eles sugerem trabalhar com o entendimento de macrovisões do que precisa ser informatizado ou automatizado e o refinamento ir acontecendo ao decorrer do projeto.

**ATENÇÃO**

Para reflexão, é interessante analisarmos o ponto de vista que Pressman e Maxim (2016, p. 85) expõem sobre a engenharia de software versus metodologias ágeis: "você não tem de escolher entre agilidade ou engenharia de software. Em vez disso, defina uma abordagem de engenharia de software que seja ágil".

Será que é possível obter as melhores práticas da engenharia de software tradicional e dos métodos ágeis para alcançar um resultado satisfatório da engenharia de requisitos do seu projeto e da entrega do seu software como um todo?

É importante enfatizar o conceito que o IEEE (2014) fornece para engenharia de requisitos: a Engenharia de Requisitos é o processo que envolve a elicitação, análise, especificação, e validação de requisitos de software, bem como a gestão de requisitos durante todo o ciclo de vida do produto de software. A seguir, serão detalhadas cada uma das **fases da Engenharia de Requisitos**:

**Elicitação**: é a fase que envolve o levantamento de informações para identificação dos requisitos de um software.

Atualmente, na maioria dos projetos, o levantamento de requisitos está vinculado à substituição de uma solução em uso ou a implementação de um processo de negócio que será viável somente com a solução demandada funcionando. Nestes dois casos, estamos longe de "informatizar o papel". A substituição de uma solução já existente é crítica e problemática. Os requisitos fornecidos podem ser totalmente vinculados à solução atual, viciados pela forma de trabalho imposta por ela. Isso significa, que as partes interessadas ao invés de fornecer requisitos de negócio originais, do problema a ser resolvido, trazem uma versão deturpada dos processos de negócio, mascarados pela solução que o sistema atual dá.

As partes interessadas podem se manter avessos à troca da solução. É uma tendência do ser humano querer se manter estável com soluções já existentes, mesmo que deficientes. Podem surgir alguns conflitos vinculados a esse apego ao software em uso.

Já no caso da implantação de um projeto que depende de uma ferramenta computacional para ser executado, o desafio é delimitar as necessidades de uma situação que ainda não existe. As partes interessadas devem ser treinadas na capacidade de abstração para permitir a discussão das soluções junto ao engenheiro de requisitos.

**Documentação**: os requisitos elicitados serão documentados seguindo um padrão. Diversas técnicas podem ser utilizadas nesta etapa e os requisitos podem ser descritos em linguagem natural ou uma linguagem estruturada.

**Validação e Negociação**: Os requisitos de software devem ser constantemente validados e negociados entre as partes envolvidas para garantir que os critérios de qualidade previamente definidos estejam sendo cumpridos.

**Gerenciamento**: O gerenciamento de requisitos é uma fase "guarda-chuva" que está presente permanentemente no processo de requisitos, com o intuito de tomar as medidas de gestão necessárias para a realização das demais fases.

## 2.1 PARTES INTERESSADAS

Partes interessadas são pessoas que constroem e consomem softwares de computador. Mesmo que atualmente existam softwares que "conversem" uns com os outros, os resultados destas integrações são necessidades de pessoas. Por isso, o *peopleware,* que são considerados as pessoas que trabalham diretamente ou indiretamente com tecnologia da informação são um dos aspectos mais importantes do desenvolvimento de software (MARCO; LISTER, 1999).

Para Pressman e Maxim (2018), as partes interessadas são quem impactam ou são impactados por um projeto de software. Os autores aiunda classificam cinco grupos que estes envolvidos podem estar enquadrados:

- Gerentes sêniores: responsáveis por definir os macro-objetivos do negócio, com influência significativa no projeto.
- Gerente de projeto (técnicos): tem a missão de planejar, motivar, organizar e controlar o trabalho da equipe de desenvolvimento do software.
- Profissionais: com as habilidades técnicas necessários para desenvolver a engenharia necessária para a criação dos produtos.
- Clientes: são os que especificam as necessidades do software.
- Usuários: São os que interagem com o software quando pronto.

O propósito da atividade de análise das partes interessadas como o trabalho de identificação das partes interessadas que podem ser afetadas por uma necessidade de negócio. Essa identificação pode ser realizada para o projeto com um todo ou por fase do projeto. É essencial que seja determinada a influência e/ou a autoridade de cada uma das partes interessadas para aprovação das entregas do projeto (IIBA, 2011).

A atividade de análise das partes interessadas é realizada a partir do momento que a necessidade de negócio é identificada, e será uma atividade contínua durante o processo de análise de requisitos (IIBA, 2011).

O primeiro passo do processo análise das partes interessadas é identificar as partes interessadas, os papéis, responsabilidades e autoridade sobre os requisitos que cada parte detém ou é afetada (IIBA, 2011).

O IIBA (2011) sugere que as partes interessadas sejam identificadas no início do projeto para que se tenha condições de garantir as entregas de requisitos em tempo hábil. Realizar a identificação das partes interessadas relevantes é uma tarefa fundamental da engenharia de requisitos e o engenheiro de requisitos deve coletar, documentar e consolidar os requisitos parcialmente conflitantes de diferentes partes interessadas (POHL; RUPP, 2011).

Pohl e Rupp (2011) relatam como consequência da falta de consideração de partes interessadas, que requisitos possam ser desconhecidos, fazendo com que novas solicitações de requisitos venham ser estabelecidas já na fase de implantação da solução. Para minimizar estas falhas, é sugerido que sejam mantidas listas das partes interessadas e que sejam atualizadas periodicamente, tornando este processo contínuo. O IIBA (2011) complementa o tema, alertando que em alguns casos requisitos descobertos podem gerar a necessidade de uma revisão completa dos requisitos de um software.

Um aspecto a se considerar em relação às partes interessadas é a complexidade. Existem dois principais aspectos que influenciam na complexidade de um grupo de partes interessadas: o número e a variedade de usuários que a parte interessada representa e o número de processos de negócio e sistemas automatizados em que ela está envolvida (IIBA, 2011).

Para o IIBA (2011), os fatores relacionados à atitude e à influência das partes interessadas devem ser considerados:

- No que tange à atitude, os aspectos centrais estão relacionados aos objetivos e metas do negócio e à abordagem da solução, à atitude com relação à colaboração e ao patrocinador.
- Já com relação à influência, é necessário conhecer a influência das partes interessadas com o intuito de viabilizar relacionamentos de confiança e com a construção destes relacionamentos, obter o comprometimento e a colaboração das partes interessadas. Alguns fatores vinculados à influência podem ser considerados: influência no projeto, influência necessária para o bem do projeto, influência na organização, influência com outras partes interessadas.

As técnicas para identificar partes interessadas podem incluir, mas não estão limitados aos seguintes itens (IIBA, 2011):

- Definição dos critérios de aceite e avaliação: é função do analista de requisitos, dentro do seu trabalho de análise das partes interessadas, identificar quais partes interessadas detém a autoridade necessária para aceitar ou rejeitar uma solução.
- Brainstorming: tem como objetivo ajudar na identificação das necessidades e requisitos que motivam as partes interessadas, bem como elaborar uma lista de possíveis papéis desempenhados por elas.
- Entrevistas: ao realizar entrevistas com partes interessadas, é muito provável que seja possível identificar outras partes interessadas que devem ser consideradas.
- Modelagem Organizacional: deve-se considerar se as unidades organizacionais ou pessoas listadas possuem necessidades e interesses únicos e que devem ser levados em conta.
- Modelagem de Processos: todas as pessoas envolvidas na execução de processos de negócios que são afetados pela solução em questão são consideradas partes interessadas. Os modelos de processos podem ser uma fonte útil para identificar as partes interessadas, especialmente se os processos relacionados forem afetados. Além disso, a categorização das partes interessadas com base nos sistemas que suportam seus processos de negócios pode ser benéfica quando mudanças são necessárias nos processos e sistemas.
- Workshops de Requisitos: durante as sessões de workshop de requisitos, o analista de requisitos pode indagar aos participantes se eles têm alguma sugestão de outras partes interessadas que devem ser consideradas.
- Análise de Riscos: existem riscos para o projeto que podem surgir devido às atitudes das partes interessadas ou devido à capacidade das principais partes interessadas de participar do projeto.
- Cenários e Casos de Uso e Histórias de Usuários: a identificação dos papéis desempenhados pelas partes interessadas pode fornecer uma compreensão mais clara de quem está envolvido e quais são suas responsabilidades e expectativas em relação à iniciativa em questão. Essa compreensão pode ajudar a identificar os atores e as funções relevantes que precisam ser considerados ao projetar e

implementar a iniciativa.

- Modelagem do Escopo: os modelos de escopo devem representar as partes interessadas que estão fora do escopo da solução, mas que irão interagir com ela de alguma forma.
- Pesquisa/Questionário: pode ser benéfico identificar características compartilhadas entre um grupo de partes interessadas.

O IIBA (2011, p. 35) apresenta a lista de partes interessadas, papéis e responsabilidades que deve conter:

> Lista de papéis, nome e cargo das partes interessadas, categoria das partes interessadas, localização das partes interessadas e formas de contato; necessidades especiais; número de indivíduos interessados neste papel; descrição da influência e interesse das partes interessadas; documentação dos níveis de autoridade das partes interessadas.

Depois de identificadas as partes interessadas, é necessário que seja realizado o gerenciamento contínuo destas partes interessadas. Em sua obra, Pressman e Maxim (2016) deixam muito claro um dos motivos pelos quais devemos nos preocupar com isso: "cada interessado tem uma visão diferente do sistema, obtém diferentes benefícios quando o sistema é desenvolvido com êxito e está sujeito a diferentes riscos caso o trabalho de desenvolvimento venha a fracassar".

Com frequência, os projetos envolvem uma quantidade grande de partes interessadas. Deste modo, é necessário realizar uma seleção criteriosa com o intuito de que a elicitação de requisitos ocorra com as partes interessadas mais adequadas (POHL; RUPP, 2011).

O processo de gerenciamento das partes interessadas exige habilidades do analista de requisitos para: lidar com a comunicação contínua com as partes interessadas; envolvê-los de forma a evitar reações contrárias a solução proposta; apresentar os benefícios da mudança proposta aos mais céticos e aos que estão confortáveis com a situação atual (POHL; RUPP, 2011).

É sugerida a aplicação uma abordagem utilizando os diversos pontos de vista das partes interessadas. Desta forma, será possível identificar os requisitos como um todo, montar uma matriz com os requisitos emergentes, que podem incluir inconsistências e/ou conflitos para que possam ser apresentados aos tomadores de decisão com o intuito de viabilizar um compêndio consistente de requisitos (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2018).

Os mapas das partes interessadas são diagramas que representam o relacionamento das partes interessadas com a solução envolvida. As duas abordagens mais comuns são a matriz nível de influência x nível de interesse e o diagrama cebola, que indica o nível que cada parte interessada está envolvida com a solução (IIBA, 2011).

## 2.2 GERENCIAMENTO E CONTROLE DE MUDANÇAS DE REQUISITOS DE SOFTWARE

Para Pressman e Maxim (2016, p.623), as mudanças fazem parte da essência do software:

> Mudanças são inevitáveis quando o software de computador é construído e podem causar confusão quando os membros de uma equipe de software estão trabalhando em um projeto. A confusão surge quando as mudanças não são analisadas antes de serem feitas, não são registradas antes de serem implementadas, não são relatadas àqueles que precisam saber ou não são controladas de maneira que melhorem a qualidade e reduzam os erros.

O IIBA (2011) determina que o propósito do processo de Gerenciamento de Requisitos é a definição do processo que será empregado para aprovar requisitos e gerenciar as mudanças no escopo dos requisitos ou da solução.

A atividade inclui a vinculação do processo para as mudanças nos requisitos, quem são as partes interessadas que tem o poder de aprovar as mudanças, quem será consultado e quem será apenas comunicado das mudanças, e até mesmo quem não precisa ser envolvido (IIBA, 2011).

O processo também faz a avaliação das necessidades de rastreabilidade dos requisitos e a determinação de quais atributos dos requisitos serão utilizados (IIBA, 2011).

Sommerville (2018) reforça que o gerenciamento de requisitos é o processo de compreender e controlar as mudanças nos requisitos de sistemas. Esse processo é realizado em conjunto com os outros processos da engenharia de requisitos. O seu planejamento tem início desde o levantamento inicial dos requisitos, ou seja, no processo de elicitação dos requisitos, a partir do momento em que um esboço da versão do documento de requisitos estiver disponível.

A partir da definição dessa linha de base do escopo de requisitos, o objetivo é obter a aprovação das partes interessadas que tem autoridade para determinar os requisitos daquele projeto. Após esta aprovação de uma linha de base, tem-se uma versão do conjunto de requisitos daquele software, e, a partir deste ponto, qualquer alteração nos requisitos deverá envolver um processo de gestão de mudanças (IIBA, 2011).

Os mesmos critérios utilizados na concepção, ou até processos mais rígidos e apurados precisam ser utilizados para gerenciar requisitos de software. Mesmo após o *go-live*, mesmo que o sistema já esteja implantado há muito tempo na organização, é necessário ter cuidado com as mudanças nos requisitos. Além de garantir que o software mudará sem perder seus objetivos principais, sem perder seu escopo e sem

desconsiderar os contextos que o levaram até ali, a gestão de mudanças é essencial para garantir que a documentação daquele software se mantenha atualizada e siga cumprindo seu papel de concentrar as informações referentes ao funcionamento daquele produto.

## 2.3 RISCOS EM REQUISITOS DE SOFTWARE

Riscos são oportunidades ou ameaças em relação a um projeto. São eventos que caso venham ocorrer podem causar impactos no contexto que estão inseridos (PMI, 2017).

Para todo o desenvolvimento de software, e mais especificamente para a engenharia de requisitos, a análise de riscos se torna fator essencial. Como já estudamos, problemas na fase de requisitos desencadeiam retrabalho em todas as fases seguintes, aumentando os custos e prazos dos projetos.

O PMI (2017) apresenta um processo estruturado para gerenciamento de riscos, que consiste na elaboração de um plano de gerenciamento dos riscos que fornece a abordagem para identificar, analisar e monitorar riscos. Este documento é estruturado para conter a estratégia em relação aos riscos, metodologia, papéis e responsabilidades, fontes de financiamento para lidar com os riscos, prazos de reavaliação dos riscos, definição das categorias dos riscos.

Os processos de planejamento em relação a riscos sugeridos pelo PMI (2017), incluem a identificação dos riscos, a realização da análise qualitativa dos riscos, a análise quantitativa e o planejamento das respostas aos riscos. O documento resultado destas atividades é o registro de riscos, que conterá a lista de riscos identificados, os prováveis impactos e as ações previstas para mitigar estes riscos.

Para Pressman e Maxim (2016), é crucial que, se forem identificados riscos de alto impacto e alta probabilidade, seja desenvolvido um plano de contingência. Além disso, o planejamento do projeto deve ser revisado para incluir a possibilidade de um ou mais desses riscos ocorrerem. É importante considerar os possíveis danos que a exposição a esses riscos pode causar ao projeto.

No que tange aos requisitos, os riscos relativos ao negócio ameaçam a viabilidade do software que está sendo criado e podem ameaçar o projeto ou o produto. Os cinco principais riscos de negócio são (PRESSMAN; MAXIM, 2016):

- risco de mercado: criar um excelente produto ou sistema que ninguém realmente quer pode ser uma experiência frustrante e dispendiosa. É importante lembrar que o sucesso de um produto ou sistema está intrinsecamente ligado a sua aceitação pelos usuários e ao valor que ele agrega às suas vidas;

- risco estratégico: elaborar um produto que não tem um alinhamento com a estratégia global de negócios da empresa;
- risco de vendas: produzir um produto que o time de vendas não tem as habilidades necessárias para vende-lo;
- risco gerencial: Perder o suporte da alta administração devido a alterações no direcionamento da organização ou na composição da equipe;
- risco de orçamento: Considerando que um projeto de software tem elevados e leva um tempo considerável, é possível que ocorra replanejamentos no orçamento da empresa que possam fazer com que o projeto perca sua fatia do orçamento e fique sem recursos para ser executado.

Considerando o cenário de desenvolvimento de software, é essencial que seja implantada uma estratégia relacionada a riscos, seja ela mais ou menos formal, mais ou menos estruturada, mas que abranja um mínimo de garantia de previsibilidade em relação aos rumos do projeto.

Neste tema, vimos que o gerenciamento e controle de mudanças e o controle dos riscos de software são aspectos fundamentais para garantir a qualidade e segurança dos produtos de software. As partes interessadas podem garantir que as entregas sejam feitas com o mais alto nível de qualidade e eficácia desde que observem e executem os processos de engenharia de requisitos de forma adequada.

O mantra em relação às partes interessadas é a colaboração. Os clientes, usuários, os desenvolvedores e as equipes de gerenciamento de projetos desempenham um papel fundamental na identificação e gestão de mudanças e riscos no processo de desenvolvimento de software. Por isso, devem agir em modo de integração e compartilhamento de conhecimentos. Eles precisam trabalhar juntos para garantir que o software atenda às expectativas dos usuários, seja seguro e eficiente e seja entregue dentro do prazo e orçamento previstos.

# RESUMO DO TÓPICO 2

**Neste tópico, você aprendeu:**

- Como os processos de engenharia de requisitos se enquadram no processo de desenvolvimento de engenharia de software.
- A importância das partes interessadas e como incluí-las e gerenciá-las para se obter sucesso no gerenciamento de requisitos.
- O contexto do gerenciamento de requisitos e controle de mudanças como uma fase permanente em relação ao software durante seu ciclo de vida.
- As técnicas para se trabalhar com riscos em projetos de software, vinculados principalmente a requisitos.

# AUTOATIVIDADE

1 As partes interessadas são essenciais para o desenvolvimento dos trabalhos de engenharia de requisitos. Com relação à seleção das partes interessadas, analise as sentenças a seguir:

I- Apenas diretores e gerentes devem ser envolvidos como partes interessadas, afinal são eles que decidem os rumos das organizações.
II- Para simplificar o trabalho, partes interessadas que apenas sofrem impactos do projeto não precisam ser envolvidas. Apenas as que causam impactos no software.
III- Quem é impacta ou é impactado pelo escopo do software a ser desenvolvido deve ser envolvido como parte interessada no projeto.
IV- Podem ser criados grupos que representam partes interessadas para organizar melhor o trabalho do projeto.

Assinale a alternativa CORRETA:

a) ( ) As sentenças I e II estão corretas.
b) ( ) Somente a sentença I está correta.
c) ( ) As sentenças I e III estão corretas.
d) ( ) As sentenças III e IV estão corretas.

2 Riscos devem ser considerados em todo projeto e, em um cenário de desenvolvimento de software que tem muitas incertezas, uma atenção especial aos riscos deve ser considerada. Com relação aos riscos no processo de engenharia de requisitos, analise as sentenças a seguir:

( ) Mudanças no ambiente organizacional não representam riscos para o desenvolvimento de software.
( ) Deve-se ser destacado um orçamento específico para mitigação de riscos para evitar que o projeto fracasse.
( ) Não é necessário definir um plano de respostas para os riscos, o mais indicado é reagir conforme eles forem acontecendo, se acontecerem.
( ) A análise qualitativa e quantitativa de cada risco identificado permite que os riscos sejam priorizados e as respostas a eles sejam planejadas adequadamente.

Assinale a alternativa CORRETA:

a) ( ) As sentenças I e III estão corretas.
b) ( ) Somente a sentença IV está correta.
c) ( ) As sentenças I e III estão corretas.
d) ( ) Somente a sentença III está correta.

3 O processo de desenvolvimento de software é iterativo e pode ser adaptado de acordo com as necessidades e feedbacks do cliente e dos usuários. O gerenciamento de requisitos é essencial para o processo de desenvolvimento de software. Com relação ao gerenciamento de requisitos, assinale a alternativa CORRETA:

a) ( ) Ocorre durante a construção do software, para que ele possa ser produzido.
b) ( ) Permanece sendo executado durante todo o ciclo de vida do sistema.
c) ( ) Não tem necessidade de ocorrer em contextos de desenvolvimento de software ágil, já que os processos ágeis são preparados para mudanças.
d) ( ) O gerenciamento de requisitos apenas registra mudanças, já que não tem como evita-las.

4 Requisitos de Software têm papel central no desenvolvimento de software e compõem parte de um projeto. De que maneira os requisitos influenciam outras atividades do processo de software? Justifique.

5 O relacionamento com e entre as partes interessadas pode impactar diretamente no trabalho de um engenheiro de requisitos. Disserte sobre estratégias que podem ajudar a obter resultados adequados com relação às partes interessadas.

# ELICITAÇÃO DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 3, abordaremos os conceitos de elicitação de requisitos. Este trabalho é essencial para o sucesso do desenvolvimento de software, e não há uma única forma de se fazê-lo, nem tão pouco uma forma mais ou menos adequada. Existe diversas formas que podem ser aplicadas de acordo com o contexto do projeto e do produto de software a ser construído.

Neste tema, conheceremos os conceitos do mapeamento de processos utilizando a notação BPMN – *Business Process Model and Notation* que permite transformar em um diagrama o funcionamento dos processos de uma organização, permitindo assim que haja mais clareza para a definição dos requisitos de software baseado num processo conhecido por todos.

Estudaremos também o *Design Thinking* que é uma técnica moderna e criativa para resolver problemas complexos e software sendo um problema complexo e criativo encoraja-se o uso para leventamento de requisitos. Trabalhamos também técnicas de levantamento de requisitos, como entrevistas, questionários, observação, análise de documentos e provocamos a reflexão sobre como combinar estas diversas técnicas de acordo com a necessidade dos projetos a serem executados, o nível de conhecimento e engajamento das partes interessadas e até o mesmo a quantidade de áreas e pessoas envolvidas.

## 2 ELICITAÇÃO DE REQUISITOS

Você já parou para pensar sobre como os sistemas nascem, como eles são concebidos? De onde surgem as necessidades para se desenvolver um sistema?

As organizações, para se manter competitivas, precisam ter sob seu domínio sistemas que permitam a gestão de todas as suas operações. Inclusive, cada vez mais os sistemas são construídos para permitir operações de negócio que anteriormente não poderiam ser executadas, sejam viáveis através de soluções de tecnologia (PLEEFGER, 2004).

As organizações precisam executar, controlar e gerenciar suas atividades, sejam elas de que natureza for. Estas organizações podem estar vinculadas a legislações, regulamentos, normas de um determinado setor. Por exemplo, uma empresa que faz

venda de produtos ao consumidor, precisa emitir as notas fiscais. As notas fiscais são regidas pela legislação tributária. Quando falamos de vendas de produtos, legislação estadual, e quando falamos de prestação de serviços, municipal. As empresas submetidas a esta necessidade precisam ter sistemas que atendam aos requisitos destes instrumentos para que possam manter regularidade com relação as suas operações.

Um outro fato gerador de necessidade de criar ou alterar sistemas já existentes é se houver alguma alteração na legislação. Por exemplo, um imposto passa a ser calculado de uma outra forma. Acontece também o tipo de demanda interna, por exemplo, se num determinado tipo de negócio passa a prestar algum tipo de serviço novo, ou precisa implantar algum tipo promoção para mantê-lo negócio atrativo no mercado. Nesses casos, o sistema vai precisar ser alterado!

Atualmente, a maioria das organizações já tem algum sistema implantado em todas as suas áreas de negócio. É raro uma empresa não ter um sistema, por mais simples que ele seja, para atender as suas áreas dentro da organização. Isto é muito reflexo da informatização que a ocorreu a partir da década de 90, e muito mais forte a partir dos anos 2000. Isso significa que a empresa tem a necessidade de evoluir esses sistemas (PLEEFGER, 2004).

Essa evolução pode ocorrer tanto por aspectos tecnológicos, quantos aspectos de negócio. Aspectos tecnológicos tem relação com a plataforma de desenvolvimento, ambiente em que esses sistemas estão rodando. Pode ser que seja necessário que um software que esteja na nuvem, e o que foi construído lá há dez, vinte anos atrás, não tem essa capacidade. Pode ser que agora seja preciso que esse software seja acessível através de dispositivos móveis e a arquitetura em que ele foi construído não atende a essa demanda.

Com relação aos aspectos de negócio, podemos exemplificar com o comércio virtual. Os sistemas das organizações precisam se adaptar, se integrar e serem naturalmente aptos a integração com os *Marketplaces*. O comércio virtual já existe há mais de uma década, mas não existia essa preferência por ele. Então, hoje, os sistemas de grandes empresas estão mais voltados para a experiência digital, para a interação híbrida, em que um consumidor pode iniciar seu contato com a marca pelo digital e encerrar sua compra no presencial, nos casos em que o consumidor faz a compra na loja virtual e retira na loja física.

Um outro modo é quando a empresa irá desenvolver um software para ser comercializado. Isso é o cenário das empresas de desenvolvimento e produção de software. A empresa não desenvolverá um software para si mesmo, para alguma área da sua organização. Ela constrói um software para ser comercializado e utilizado por diversas empresas.

Geralmente, as empresas desenvolvedoras têm o interesse de atender uma determinada área de negócio com um software. Por exemplo, a empresa de desenvolvimento ABC planeja construir um software para gerenciar a folha de pagamento para micro e pequenas empresas. Então, o que ela desenvolve é um produto. Para produzir este software, ela vai pesquisar o mercado, vai pesquisar como funciona um setor de folha de pagamento de uma empresa naquele determinado nicho, vai pesquisar os detalhes de como a maioria das empresas está trabalhando com isso, fará uma pesquisa de usabilidade, com alguns testes, ensaios e, de repente, lançar um MVP, que é um produto mínimo viável, para que possa validar se sua solução atende às necessidades dos usuários.

Independentemente de como surgiu a necessidade, se ela foi uma demanda externa, se ela foi uma resposta a provocação de concorrente, se ela foi uma legislação, se foi uma forma que aquele negócio evoluiu ou uma necessidade tecnológica, como pode ser entendido o que que precisa ser desenvolvido?

Vamos supor que a equipe de desenvolvimento tem todas as habilidades de arquitetura de software, de banco de dados, de estruturação e de infraestrutura para poder colocar o nosso sistema numa plataforma escalável, mas isso tudo adianta se a equipe não sabe o que precisa desenvolver? Do que vale a equipe ter todas as habilidades técnicas no que envolve a programação de desenvolvimento, de código fonte e de construção de software se ela não sabe exatamente que problema será resolvido?

É necessário considerar que o desenvolvimento de software é uma área viabilizadora, no meio de negócios. A área de desenvolvimento de software não existe por si só.

É importante ressaltar que o nosso usuário ele é fonte de informação, mas ele não vai saber tudo sobre aquela área de negócios, sobre aquele sistema. Ele não precisa e não saberá tudo, mas com técnicas que serão estudadas nessa disciplina será possível extrair e combinar com outras técnicas, como pesquisa de mercado, pesquisa de concorrência, pesquisas nos regulamentos instituições e setoriais, nas legislações daquela área de como aquela área funciona para poder definir uma melhor solução.

Atualmente, os usuários têm um nível de conhecimento muito alto. Eles são usuários avançados de uma série de sistemas, começando pelos softwares pessoais, como aplicativos de *Internet Banking*, aplicativos de trocas de mensagens e redes sociais. Muitas vezes, o usuário acredita que ele tem como contribuir trazendo já a solução para um problema que ele enxergou dentro da organização dele. Ele vem sugerindo um desenho de tela, sugerindo onde clicar. Isso não é saudável para o processo de levantamento de requisitos, pois o cliente está focado na solução e não no problema.

Pode parecer estranho dizer que alguém que está trazendo uma sugestão está atrapalhando, e que isso não é saudável para o processo de desenvolvimento de software. Nosso papel é conscientizar os usuários. Eles precisam ter de noção de

que a responsabilidade deles é trazer os problemas, as necessidades e depois validar as soluções propostas pela equipe de desenvolvimento. Quando o usuário já traz uma solução daquele problema, existe uma tendência de ele não mostrar o problema completo. A solução trazida por ele até pode ser eficaz, mas focada em um viés daquela situação. Se ele já traz uma solução, o problema pode ficar maquiado e muitas das vezes quando a equipe de requisitos o avalia mais profundamente, consegue uma outra forma de resolver, com uma cobertura maior de situações, trazendo mais benefício para todo processo.

A ideia é interpretar bem o que o cliente traz. Se o usuário trouxer o problema e já trouxer uma solução sugerida, a equipe deve ouvir, documentar e armazenar aquela solução para ser usada apenas num segundo momento, para verificar se encaixa como parte da solução do problema, sem perder o foco em identificar qual é o problema verdadeiro, qual a necessidade de negócios aquela organização está precisando definir.

## 2.1 MAPEAMENTO DE PROCESSOS UTILIZANDO BPMN

Para que possamos conhecer o ambiente de negócios em que iremos construir o software, nos embasamos no conhecimento organizacional. As informações repassadas pelas partes interessadas daquele software são essenciais para o sucesso dele.

Temos uma garantia que as partes interessadas têm conhecimento total de como aquela empresa funciona? Muitas das vezes, o cliente vai trazer o seu problema, ele vai detalhar todas as informações que precisa, só que muitas vezes será apresentado o cenário de como é fluxo de informações, os fatos e os eventos que ocorrem naquela organização. Não é bem o cenário como as coisas funcionam na prática. No papel está escrito de certa forma, mas na prática aquela organização está executando de outra?

Os usuários, às vezes, não declaram exatamente como que as situações realmente ocorrem na prática, pois isso pode demonstrar deficiências. Imagine um processo de levantamento de um software, em que os usuários estão participando de uma série de reuniões, junto aos seus supervisores e diretores, e eles têm que declarar para a equipe que estão fazendo o levantamento de requisitos. Tal processo deveria ocorrer como algo que ele faz no dia a dia, deveria ser de um jeito, mas ele faz de outro.

Então, muitas vezes, não temos informações completas, se tornando um problema na hora que esse software fica pronto e se inicia o uso. O software é desenvolvido corretamente, mas no dia a dia da organização, muitas vezes, não é executado dessa forma. Como resolver este problema?

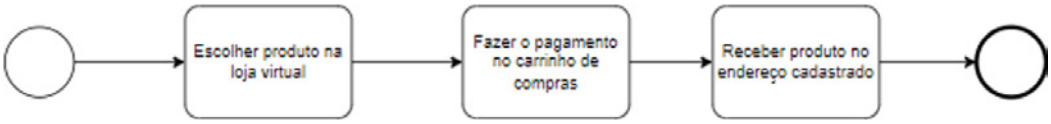
É importante ter algum mecanismo para poder entender como essas empresas funcionam. Muitas vezes, os usuários estão trazendo um processo, uma forma de trabalhar que não é a forma mais adequada para aquela organização, mas sim a forma

que o software que existe hoje impõe. Por isso tudo, é necessário entender os processos da organização para que possa ser realizado um trabalho de levantamento de requisitos completo.

Um processo dentro de uma organização é um processamento de alguma atividade de negócio. É a execução de uma sequência de atividades com um objetivo específico. Ao final de um processo, sempre teremos um resultado (CAMPOS, 2014).

Na Figura 3, podemos verificar a representação de um processo de compra realizado em uma loja on-line. O processo inicia-se com a escolha do produto, com o posterior pagamento no carrinho de compras. Por fim, recebe-se o produto no endereço cadastrado e o processo é encerrado.

Figura 3 – Processo de Compra On-line




Fonte: o autor

Para que seja possível representar estes processos utilizando um padrão, utiliza-se a BPMN, que significa *Business Process Model and Notation*. Trata-se de uma representação gráfica através de uma linguagem, de modo a representar o fluxo de um processo, possibilitando, assim, o entendimento dele (CAMPOS, 2014).

A notação BPMN possibilita que sejam desenhadas e divulgadas todas as tarefas operacionais de um negócio de forma lógica e estruturada. Também é possível apresentar os papéis envolvidos naquele processo, os eventos ocorridos e os demais componentes interligados, como entradas e saídas dos processos (OMG, 2011).

Os principais elementos de BPMN são atividades, gateways, eventos, conectores e as raias que estruturam o processo. A seguir, são detalhados cada um destes elementos (OMG, 2011).

Atividades: as atividades são divididas em tarefas e subprocessos. O ícone de atividade representa uma ação que pode ser realizada por uma pessoa ou por um sistema. Expressas por retângulos com bordas arredondadas, as atividades são os passos ocorridos durante a rotina do processo. Uma atividade é o item de trabalho com a menor granularidade, podendo ser executada por sistemas ou por indivíduos.

Gateways: os gateways são os elementos que permitir as divisões no fluxo de processos. Eles permitem condicionar, ou seja, determinar caminhos diferentes no fluxo do processo se determinada condição for ou não atendida. Os gateways, que na notação BPMN são simbolizados por losangos, representam a convergência ou a divergência da continuidade de um fluxo.

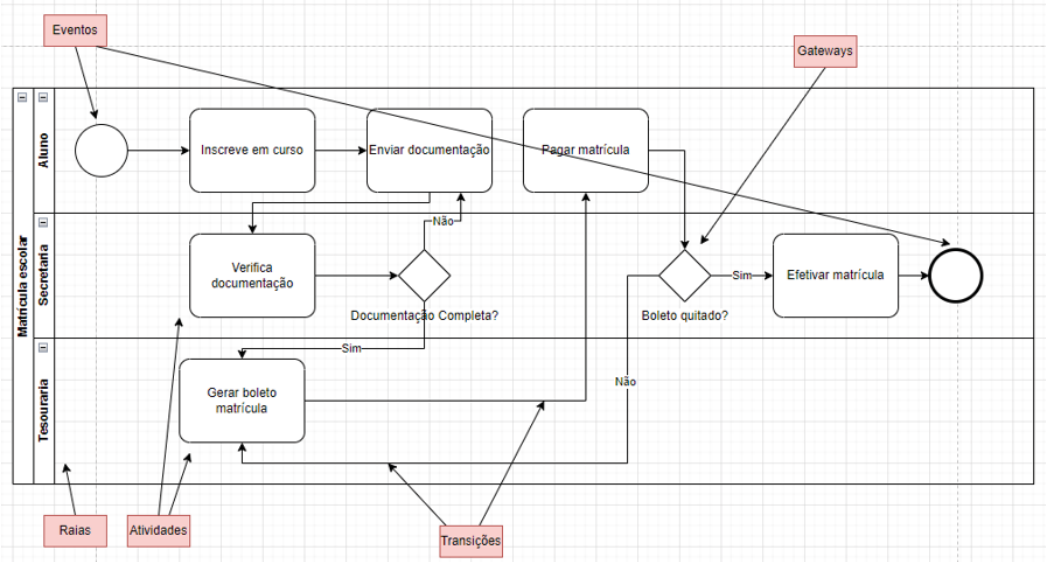
Eventos: um evento é algo que ocorre durante o andamento de um Processo. Esses eventos afetam o fluxo do processo e geralmente têm uma causa ou um impacto. O termo evento é genérico o suficiente para abranger muitas coisas em um processo. O BPMN restringiu o uso de Eventos para incluir apenas os tipos de Eventos que afetam a sequência ou tempo das atividades de um processo.

Conectores: são os elementos de ligação. Graficamente, são flechas de transição que ligam um componente ao outro. Basicamente, os objetos de fluxo necessitam ser interligados entre si, e esse papel é feito pelos objetos de conexão.

Raias: Uma raia é uma subpartição dentro de um Processo, geralmente utilizado para identificar os papéis que são os responsáveis pelas atividades que estão contidas naquela raia.

A Figura 4 apresenta os elementos listados. O processo está estruturado em três raias (Aluno, Secretaria e Tesouraria) e se inicia por meio do símbolo do evento de início, por meio do aluno. O aluno executa a tarefa de se inscrever em um curso e na tarefa seguinte, que é enviar a documentação para a Secretaria. A secretaria executa a tarefa de verificar a documentação, que direciona para o gateway (decisão) se a documentação está completa ou não. A documentação estando completa, a transição é direcionada para a raia da Tesouraria, que irá executar a atividade gerar o boleto de matrícula. O boleto é encaminhado ao aluno, que deverá executar a atividade pagar matrícula. A secretaria verifica se o boleto foi quitado, caso positivo, efetiva a matrícula e encerra o processo. Caso o boleto não tenha sido quitado, encaminha para a tesouraria gerar novo boleto, até que seja pago pelo aluno.

Figura 4 – Exemplo de um processo de Matrícula utilizando a notação BPMN




Fonte: o autor

Observe que o texto explicativo do processo é muito mais difícil de compreender do que o desenho resultado da utilização da notação BPMN. Desta forma, fica muito claro reconhecer que o BPMN é uma ótima ferramenta para mapeamento de processos, permitindo que as organizações possam conhecer melhor seus processos e facilitar o trabalho de engenharia de requisitos.

Os trabalhos com relação à descoberta de requisitos devem combinar várias técnicas e abordagens. O BPMN (Business Process Model and Notation) pode ser combinado com outras técnicas, como a técnica denominada Design Thinking. Ambas são abordagens que buscam melhorar a eficiência e a inovação nos processos empresariais. Essas metodologias têm como objetivo aprimorar a maneira como as empresas pensam sobre seus processos e como os executam. Enquanto o BPMN é um framework para modelagem de processos, o Design Thinking é uma abordagem de resolução de problemas que tem como foco o usuário final, que você conhecerá melhor a seguir.

## 2.2 DESIGN THINKING

O *design thinking* é uma forma de pensamento criativo que procura soluções para resolver problemas complexos. É uma abordagem que busca envolver papéis com diferentes perspectivas para entender profundamente o público-alvo, suas dores, necessidades e comportamentos. Portanto, as chances de produzir uma solução mais inovadora, criativa e eficiente são maiores.

O *design thinking* é fundamentado na habilidade humana de ser intuitivo, identificar padrões, gerar conceitos que possuam um significado emocional para além da funcionalidade, e expressar ideias por meio de formatos distintos dos tradicionais símbolos e palavras (BROWN, 2018).

É importante mencionar que o design vai muito além do formato e estética de produtos. O design compõe, além destes fatores, a experiência dos usuários daquele produto/serviço. O *design thinking* entra como a força da abordagem centrada na jornada dos usuários dos produtos e serviços envolvidos (BROWN, 2018).

O pilar inicial do *design thinking* é a inspiração, uma fase que comtempla um entendimento inicial do problema e estudo das possibilidades envolvidas. O outro pilar é da empatia, que faz se colocar no lugar do outro e compreender o problema a partir de diversos pontos de vista. Já a criatividade é o pilar para se inspirar e produzir ideias criativas a partir das experiências geradas nas etapas. Um outro ponto que é bastante valorizado no *design thinking* é o aprendizado com os erros. Aceita-se que os erros são inevitáveis e que, se aconteceram, devemos colher os melhores conhecimentos deles praticando a melhoria contínua (MELLO; ALMEIDA NETO; PETRILLO, 2021).

A seguir, serão apresentadas as etapas do *design thinking*. Elas são um tipo de roteiro a ser seguido para aplicá-lo, mas é importante salientar que não é uma regra totalmente fechada. O trabalho pode ser organizado de outras formas, de acordo com as necessidades do projeto específico (BROWN, 2018).

- Imersão: a fase de entendimento se inicia com o processo de compreender a si mesmo, utilizando ferramentas como a análise SWOT, que avalia as forças, fraquezas, oportunidades e ameaças da empresa que afetam o projeto. Antes desse processo, é discutido em detalhes o problema enfrentado e são definidos o escopo e os limites do projeto. Durante a imersão, é gerado muito material de estudo que deve considerar diversos pontos de vista e perspectivas, utilizando pesquisas exploratórias e referências. Após essa etapa inicial, avança-se para a imersão profunda por meio de entrevistas e trabalhos de campo, os quais irão gerar gatilhos para as próximas fases.

- Análise e Síntese: durante a fase de imersão, é comum surgir uma grande quantidade de conteúdo, porém, nem tudo o que é produzido será útil. Na etapa de análise e síntese, é realizada uma avaliação lógica e racional dos insights gerados anteriormente, a fim de identificar padrões e categorizar as ideias. Representações gráficas, como diagramas e mapas conceituais, podem ser úteis para uma melhor visualização do problema. Nesse momento, os itens discutidos são filtrados e o projeto começa a ser desenvolvido com a definição de um público-alvo e de uma estratégia de trabalho.

- Ideação: nesta fase, a equipe de trabalho começa a buscar soluções para o problema apresentado, tendo em mente que nenhuma solução é perfeita. É importante estar ciente de que surgirão diversas ideias, todas baseadas nas necessidades identificadas nas fases anteriores. É encorajado o uso de métodos para explorar a criatividade dos participantes e estimular o pensamento criativo. As ideias mais audaciosas são construídas em um ambiente livre de julgamentos. As equipes precisam ter liberdade para experimentar e errar. O incentivo à experimentação permite que novas soluções surjam e possam trazer resultados inovadores.

- Prototipação ou Prototipagem: pesquisas indicam que a maioria das empresas enfrenta falhas no lançamento de produtos e serviços no mercado, e isso muitas vezes não é causado por negligência, mas sim pela complexidade das variáveis envolvidas. No entanto, a adoção do design thinking pode ajudar a superar esse medo de assumir riscos e usar a experimentação como um diferencial. Para isso, a ideia do protótipo surge como uma forma de criar um Mínimo Produto Viável (MVP), que consiste em uma versão simplificada do produto que pode ser produzida rapidamente e com poucos recursos para ser testada na prática. O objetivo é sentir como a solução se comporta no mundo real antes de investir grandes quantidades de recursos em sua produção final. Para saber mais sobre o MVP, recomendamos a leitura complementar desta unidade.

- Implementação Final: o objetivo do MVP é capturar a essência da solução e testá-la na prática. Se o MVP não tiver uma boa aderência ou resultado, isso pode ser um aprendizado valioso que evita gastar recursos na construção completa e no lançamento final do produto. Se a solução for validada, a próxima etapa é a implementação, na qual são feitos os ajustes necessários para o lançamento aos usuários finais.

Considerando os aspectos dinâmicos relacionados à atividade de desenvolvimento de software, o *design thinking* pode ser uma técnica a ser utilizada, combinada com outras técnicas de mapeamento de processos e de levantamento de requisitos, para se obter um resultado mais adequado, principalmente focado na resolução dos problemas das partes interessadas.

## 2.3 TÉCNICAS DE LEVANTAMENTO DE REQUISITOS

Levantamento ou elicitação de requisitos é a atividade de identificar quais são os requisitos de negócio vinculados àquele software que será construído no projeto. O termo elicitação tem o sentido de criar uma lista de requisitos como resultado de um processo de levantamento de requisitos (PRESSMAN; MAXIM, 2018).

O intuito é conhecer os objetivos, as necessidades de negócio para o software que está sendo projetado. É o primeiro passo para idealização do software. Em sequência, é importante realizar a análise e negociação desses requisitos, quando eles serão classificados junto às partes interessadas e também determinar como eles devem ser organizados de acordo com a importância para que o desenvolvimento e a liberação ocorram de acordo com as necessidades do cliente.

Supomos que foi realizado um levantamento de requisitos para um determinado software e foram identificados mais de 50 requisitos. Sabemos que terão requisitos que tem um tamanho maior, outros menores. Nesse contexto, tamanho significa o nível de complexidade e como aquele requisito impactará no contexto do software que está sendo construído. Sabemos que 50 requisitos é um número bem elevado de itens, então, o analista precisa, em conjunto com o cliente, priorizá-los, entender como eles se relacionam, qual que é a ordem lógica que eles têm. Por exemplo, em um levantamento de requisitos de um sistema de vendas, um dos requisitos é "permitir que um vendedor registre suas vendas". Trata-se de um requisito muito importante para um sistema como esse, mas para realizar uma venda é preciso, primeiro, ter os produtos que serão vendidos cadastrados. Então, primeiro, é necessário montar um requisito "permitir o cadastro de produtos e serviços", mantendo a coerência e dependência das necessidades que aquele software precisa abranger.

É muito importante ter cuidado na hora da escrita dos requisitos, pois a língua portuguesa é cheia ambiguidades e armadilhas, em que uma vírgula pode mudar todo o sentido de uma frase, podem existir interpretações diferentes etc. Em análise de sistemas, temos que escrever de uma forma que, independente do leitor, a interpretação seja a mesma.

Para o processo de levantamento de requisitos, existem algumas técnicas, e, a partir de agora, iremos falar sobre cada uma delas, como se relacionam e complementam.

A compreensão do domínio do problema é o entendimento daquela área de negócio que o sistema abrange. Então, por exemplo, será realizado o levantamento de requisitos de um sistema de um supermercado. Um analista foi destinado para este trabalho, e é essencial que ele pesquise e entenda sobre o funcionamento de um supermercado. Pode-se pensar: todos durante sua vida já estiveram num supermercado. Claro, mas num papel bem específico, papel de cliente. É preciso entender as rotinas, os processos que fazem um supermercado funcionar. Para isso, o analista de requisitos deve entender o contexto que aquela organização e aquele sistema estão inseridos. Pesquisar normas, legislações, práticas de mercado.

Existem algumas situações que podem ocorrer ao utilizar a técnica de compreensão de domínio do problema que precisam ser consideradas, como:

- Identificação de riscos: a técnica de compreensão do domínio do problema pode ajudar a identificar riscos que precisam ser gerenciados na solução do problema (PRESSMAN; MAXIM, 2016).
- Sobrecarga de informações: ao se aprofundar na compreensão do domínio do problema, é possível coletar uma grande quantidade de informações, o que pode dificultar a identificação das informações mais relevantes (IIBA, 2011).
- Viés cognitivo: A compreensão do domínio do problema pode ser influenciada por viés cognitivo, ou seja, a tendência de interpretar informações de maneira enviesada e subjetiva de acordo com experiências e conhecimentos prévios *(MOHANANI et al., 2020)*.
- Falta de tempo: a técnica de compreensão do domínio do problema pode exigir um tempo considerável para ser realizada, o que pode ser um problema em situações em que o tempo é limitado (IIBA, 2011).

Quando este profissional for aprofundar seu trabalho de levantamento de requisitos, estará muito mais preparado para conversar com as partes interessadas, já que conhece os principais processos, termos, jargões, utilizados naquela área, facilitando assim o entendimento das necessidades.

Um outro meio de se aprofundar no assunto do sistema é realizar uma análise de sistemas concorrentes, o popular *benchmark*. Esta prática visa conhecer o que os principais sistemas que atendem aquele tipo de necessidade, como eles funcionam, que tipo de escopo eles cobrem. Com as facilidades atuais, é possível ver vídeos no YouTube com treinamentos sobre o sistema de uma grande empresa que produz software para aquele tipo de negócio. Em alguns casos, pode-se conseguir o manual do sistema, já que muitos deles permitem que seja utilizado por um período de demonstração.

Ao realizar um benchmark em um processo de levantamento de requisitos, várias situações podem ocorrer, incluindo (IIBA, 2011):

- Identificação de lacunas de desempenho: o benchmark pode revelar lacunas de desempenho em relação a outros produtos ou serviços similares no mercado, permitindo que a equipe de requisitos identifique áreas onde o novo produto ou serviço precisa melhorar.
- Identificação de melhores práticas: o benchmark pode ajudar a equipe de requisitos a identificar as melhores práticas do setor, que podem ser incorporadas no novo produto ou serviço para melhorar a sua qualidade.
- Identificação de recursos essenciais: o benchmark pode ajudar a equipe de requisitos a entender os recursos necessários para criar um produto ou serviço competitivo, como orçamento, tempo e habilidades de desenvolvimento.
- Identificação de possíveis riscos: o benchmark pode ajudar a equipe de requisitos a identificar possíveis riscos, como custos de desenvolvimento mais elevados ou dificuldades técnicas, que podem afetar a viabilidade do produto ou serviço.

- Estabelecimento de metas e critérios de sucesso: o benchmark pode ajudar a equipe de requisitos a estabelecer metas e critérios de sucesso claros para o novo produto ou serviço, com base no desempenho de outros produtos ou serviços similares no mercado.
- Conhecimento da concorrência: o benchmark pode revelar a existência de concorrentes e produtos similares que possam competir com o novo produto ou serviço, exigindo que a equipe de requisitos faça ajustes no plano para garantir a viabilidade e a competitividade do produto ou serviço.
- Complexidade excessiva: o benchmark pode levar a equipe de requisitos a tentar incorporar muitos recursos ou funcionalidades ao produto ou serviço, levando a uma complexidade excessiva e, potencialmente, a um produto ou serviço difícil de usar e manter.

Desta forma, o analista de requisitos vai começando a compreender os termos nos temas daquela área que ele está informatizando. É muito comum que analistas de requisitos que trabalham muito tempo com determinado tipo de sistema se tornem especialistas naquele tipo de negócio. Existem analistas de requisitos focados na área contábil, na área jurídica, na área financeira, na área educacional. entre outros. A experiência maior em tipo de domínio de problema faz com que os trabalhos se tornem mais fáceis e a chance de dar certo aumente. Nada impede que um analista de requisitos "genérico" aprenda sobre um novo tipo de negócio que ele nunca trabalhou, afinal, todos tiveram sua primeira vez.

## IMPORTANTE

Para facilitar os processos de engenharia de requisitos, a ferramenta glossário pode ser utilizada. Um glossário é um documento que contém uma lista de termos técnicos e de negócios que são usados durante o processo de engenharia de requisitos. O objetivo do glossário é estabelecer uma terminologia comum para todos os membros da equipe do projeto e para os interessados no projeto.

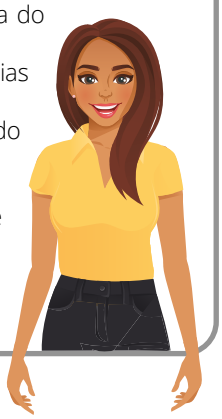
No contexto da engenharia de requisitos, o glossário é uma ferramenta importante para garantir que todas as partes interessadas entendam os termos e conceitos técnicos usados no projeto de software. Ele ajuda a reduzir a ambiguidade e a inconsistência na comunicação entre os membros da equipe e entre a equipe e os stakeholders do projeto.

O glossário normalmente inclui uma definição clara e concisa de cada termo técnico ou de negócios relevante para o projeto. A definição deve ser concisa e fácil de entender para garantir que todos os membros da equipe entendam os termos da mesma maneira. Também pode incluir acrônimos, abreviações e siglas que são usados no projeto.

O glossário é atualizado ao longo do ciclo de vida do projeto, à medida que novos termos são introduzidos ou definições existentes são modificadas ou atualizadas. Ele é uma ferramenta valiosa para melhorar a comunicação e garantir que todos os membros da equipe estejam falando a mesma linguagem, o que ajuda a evitar mal-entendidos e erros durante o processo de engenharia de requisitos.

Para ficar mais claro, a seguir é apresentado um exemplo de um glossário de um sistema hospitalar:

Admissão: processo de registro do paciente no hospital.
Alta: liberação do paciente do hospital após tratamento.
Anamnese: entrevista clínica realizada pelo médico para coleta de informações sobre o histórico médico do paciente.
Centro Cirúrgico: espaço destinado a procedimentos cirúrgicos.
Classificação de Risco: processo de avaliação do nível de urgência do paciente.
Consulta: encontro entre médico e paciente para diagnóstico e tratamento.
Diagnóstico: identificação da doença ou condição médica do paciente.
Enfermaria: ala destinada ao atendimento de pacientes hospitalizados.
Farmácia Hospitalar: setor responsável pelo fornecimento de medicamentos.
Internação: processo de hospitalização do paciente.
Laudo Médico: documento que registra os resultados da avaliação médica.
Leito: cama hospitalar onde o paciente fica internado.
Prontuário Médico: documento que contém o histórico médico do paciente.
Receituário: documento que contém a prescrição de medicamentos.
Reserva de Leito: processo de reserva de um leito para internação futura do paciente.
Sala de Emergência: espaço destinado ao atendimento de emergências médicas.
Terapia Intensiva: unidade destinada ao atendimento de pacientes em estado grave.
Triagem: processo de avaliação inicial do paciente.
Unidade de Terapia Intensiva (UTI): espaço destinado ao atendimento de pacientes graves e críticos.
Visita: encontro entre paciente e familiares ou amigos durante a hospitalização.



Com um trabalho de entendimento do domínio do problema, de análise de sistemas concorrentes, pode-se iniciar a coleta de requisitos. Neste processo, o analista de requisitos interage com as partes interessadas para entender, discutir e documentar as necessidades em relação ao sistema. Quando os requisitos vão ser coletados, o entendimento da análise de domínio do problema do analista de requisitos é apurado e potencializado. Este processo de coleta envolve uma aproximação com os usuários, afinal, se for perguntado quais os requisitos para o sistema, eles não saberão dizer. A ideia é ir para o seguinte lado: que problema o usuário pretende resolver com aquele sistema? Como que a empresa trabalha com isso atualmente? Podemos mesclar um trabalho de mapeamento de processos junto a esta habilidade de levantar requisitos. Com isso, tem-se, no final, o processo mapeado e os requisitos também levantados.

Algumas perguntas comuns a se fazer para iniciar um processo de levantamento de requisitos com usuários são (PRESSMAN; MAXIM, 2016):

- Como é o dia a dia do trabalho da área que faz parte?
- Qual que é o problema que pretende resolver com o novo sistema?
- Qual é a maior dificuldade da área atualmente?

- Existem algum processo que existe um desperdício de tempo, algo que poderia fazer mais rápido?
- Que trabalhamos manuais/operacionais são feitos hoje e existem oportunidades que um sistema auxilie?
- Qual o tipo de informação é utilizada como entrada e como saída dos processos executados pela área?
- Que tipo de indicadores são controlados como fatores de sucesso/fracasso da área?

Com um pouco de compreensão do problema, o analista de requisitos consegue fazer essas perguntas e obter respostas muito valiosas, que levarão a outras perguntas. É um trabalho bem interativo. Um analista de requisitos buscar estressar todo o entendimento sobre uma determinada situação, mas isso, muitas vezes, vai acontecer de forma gradativa, pois seu entendimento vai aumentando cada vez mais com que o processo de levantamento de requisitos vai avançando (SOMMERVILLE, 2018).

Os usuários eles podem ser abordados de várias formas pelos analistas de requisitos para fazer este processo de levantamento. Vai depender do tipo do projeto, do acesso que se tem as partes interessadas, dependendo também do tipo de sistema e da quantidade de usuários. Mesmo que exista um grupo de usuários muito grande, por exemplo, uma equipe que irá fazer um novo software frente de caixa ou para parte de caixa, financeiro, de crediário do de uma grande varejista brasileiro, que possua mais de 1000 lojas em todo Brasil, não será possível num processo de levantamento de requisitos falarmos com todos os usuários que são caixas de todas as lojas do Brasil. Podemos falar com alguém que represente esse tipo de usuário e possa definir as necessidades daquele público do software.

Neste tipo de cenário, o conceito de usuários-chave pode ser aplicado, que são pessoas que definem os processos de uma área específica. Em alguns modelos de levantamento, são escolhidas pessoas que representam o restante da população de usuários, por exemplo, uma pessoa que trabalha na função de caixa de loja de cada estado da federação, desta forma, trazendo representatividade e respeitando particularidades, pois pode ser que em alguns estados alguns processos daquela área são diferentes por particularidades locais, que precisarão ser respeitadas em um futuro sistema. Com relação ao tema, Pressman e Maxim (2016, p. 144) apresentam uma observação muito pertinente: "se um sistema ou produto vai atender a muitos usuários, esteja absolutamente certo de que os requisitos foram extraídos de uma amostra representativa deles. Se apenas um usuário definiu todos os requisitos, o risco de não aceitação é grande".

Um dos meios de realizar um levantamento de requisitos é através da aplicação de um questionário. Num questionário, são elaboradas várias perguntas, como num formulário de pesquisa, e encaminhadas para os usuários. Estas perguntas devem-se basear em informações pré-conhecidas sobre as necessidades do software, como análise do domínio do problema e de softwares concorrentes. Pode-se restringir a um

grupo específico ou uma população maior. Quanto mais pessoas, mais fechado deve ser o questionário, limitando a criatividade e tornando o processo mais restrito. O motivo é que se o questionário possuir perguntas com respostas livres, com um número de respondentes muito grande, o trabalho de análise destas respostas será enorme (IIBA, 2011; SOMMERVILLE, 2018; PRESSMAN; MAXIM, 2016)

Exemplos de perguntas genéricas que podem ser feitas num questionário clássico (PRESSMAN; MAXIM, 2016):

- Qual é o principal objetivo do software que você precisa?
- Quem serão os usuários finais do software?
- Qual é o problema que o software deve resolver?
- Qual é o processo atual utilizado para resolver o problema?
- Quais são as funcionalidades essenciais que o software deve ter?
- Quais funcionalidades adicionais seriam desejáveis para atender as suas necessidades?
- Que tipo de dados precisa ser armazenado e gerenciado pelo software?
- Como você espera que o software seja integrado a outros sistemas que você usa atualmente?
- Qual é o prazo para o desenvolvimento e implementação do software?
- Quais são os principais requisitos de segurança que o software deve atender?

Essas perguntas ajudam a entender as necessidades e expectativas dos usuários em relação ao software, os problemas que precisam ser resolvidos e as funcionalidades que são mais importantes para o sucesso do projeto. Estas perguntas servem como um ponto de partida para o levantamento de requisitos, mas devem variar e se tornar mais específicas de acordo com o contexto de cada projeto, de cada organização, de cada grupo de partes interessadas.

Num exemplo de um levantamento de requisitos de um Pet Shop, as perguntas do questionário poderiam incluir:

- Qual é o tamanho do seu petshop e quantos funcionários você tem?
- Quais são os principais serviços que o petshop oferece?
- Como você gerencia o agendamento de serviços atualmente?
- Como você controla o estoque de produtos (por exemplo, rações, brinquedos, acessórios) e o que precisa ser comprado novamente?
- Como você gerencia o registro e o histórico de seus clientes e animais de estimação?
- Quais são as principais dificuldades que você enfrenta na gestão do petshop?
- Como você acompanha o desempenho financeiro do petshop (por exemplo, fluxo de caixa, faturamento, lucro)?
- Quais são as principais métricas de desempenho que você gostaria de ver no software (por exemplo, número de agendamentos, produtos mais vendidos, taxa de ocupação do banho e tosa)?

- Como você espera que o software ajude a aumentar a eficiência operacional do petshop?

Lembrando da recomendação de que, em alguns cenários, é mais produtivo ou recomendado ter um questionário mais fechado, para que se possa avaliar um cenário objetivo, segue um exemplo do mesmo questionário anterior, com o exemplo de opções fechadas.

Qual é o tamanho do seu petshop?
a) Pequeno (até 5 funcionários)
b) Médio (de 6 a 15 funcionários)
c) Grande (mais de 15 funcionários)

Quais são os principais serviços que o petshop oferece?
a) Banho e tosa
b) Veterinária
c) Petshop (venda de produtos para animais de estimação)
d) Hospedagem

Como você gerencia o agendamento de serviços atualmente?
a) Livro de agendamento físico
b) Agenda eletrônica (como o Google Agenda)
c) Software específico de agendamento

Como você controla o estoque de produtos?
a) Controle manual (por exemplo, planilha)
b) Software específico de gerenciamento de estoque
c) Não controla o estoque

Como você gerencia o registro e o histórico de seus clientes e animais de estimação?
a) Ficha de papel
b) Planilha eletrônica
c) Software específico de gestão de clientes e animais de estimação

Qual é a maior dificuldade que você enfrenta na gestão do petshop?
a) Gerenciamento de estoque
b) Controle de agendamentos
c) Gestão financeira
d) Gerenciamento de clientes e animais de estimação
e) Outro (especifique): _____

Qual é o seu principal objetivo ao usar o software de gerenciamento do petshop?
a) Aumentar a eficiência operacional
b) Melhorar a experiência do cliente

c) Aumentar a lucratividade do negócio
d) Outro (especifique): _____

Como você espera que o software ajude a aumentar a eficiência operacional do petshop?

a) Automatizando tarefas manuais
b) Melhorando o gerenciamento de agendamentos
c) Reduzindo erros e retrabalho
d) Outro (especifique): _____

Qual é a principal métrica de desempenho que você gostaria de ver no software?

a) Número de agendamentos
b) Produtos mais vendidos
c) Taxa de ocupação do banho e tosa
d) Faturamento mensal
e) Outra (especifique): _____

Ao aplicar questionários de levantamento de requisitos, algumas situações que podem ocorrer incluem:

- Perguntas mal formuladas: se as perguntas do questionário não forem bem formuladas, podem levar a respostas imprecisas ou incompletas dos respondentes.
- Falta de respostas: os respondentes podem optar por não responder a algumas das perguntas, deixando lacunas nas informações fornecidas.
- Respostas inconsistentes: pode haver respostas inconsistentes ou contraditórias entre as diferentes perguntas do questionário, o que pode levar a dificuldades na definição dos requisitos.
- Viés dos respondentes: os respondentes podem ter seus próprios pontos de vista e opiniões, o que pode levar a uma visão limitada ou distorcida dos requisitos.
- Número insuficiente de respondentes: se o número de respondentes for insuficiente, pode haver um risco de que as informações coletadas não representem adequadamente todas as necessidades e expectativas das partes interessadas.
- Prazos apertados: se o tempo disponível para a realização do questionário for limitado, pode haver respostas apressadas ou incompletas dos respondentes.
- Questões técnicas: as questões técnicas do questionário podem não ser compreendidas adequadamente pelos respondentes, levando a respostas incorretas ou incompletas.

Uma outra técnica que pode ser utilizada para levantamento de requisitos é a da entrevista. Entrevistas de levantamento de requisitos são uma técnica de coleta de informações muito comum e útil no desenvolvimento de software. Elas permitem que os desenvolvedores entendam as necessidades e expectativas dos usuários em relação ao software e, assim, possam criar soluções que atendam as suas necessidades (SOMMERVILLE, 2018).

São estruturadas agendas com sessões de entrevistas com um grupo de usuários e se discutem as necessidades/escopo do sistema. Em muitos cenários, é interessante produzir uma lista de perguntas, como se fosse um roteiro para esta entrevista. Desta forma, ajuda com que os usuários mantenham o foco no escopo definido e torna as reuniões produtivas. Em outros cenários, as entrevistas podem ser não estruturadas, em que o entrevistador se concentra em um tópico geral e permite que o entrevistado forneça informações de forma livre (PRESSMAN; MAXIM, 2016).

Sommerville (2018) destaca em sua obra que a entrevista é uma técnica poderosa, pois permite que os desenvolvedores obtenham informações sobre as necessidades do usuário em um nível mais profundo do que outras técnicas de coleta de requisitos. As entrevistas também permitem que os desenvolvedores façam perguntas de acompanhamento para esclarecer informações e obter exemplos mais precisos das partes interessadas entrevistadas.

Ao realizar entrevistas de levantamento de requisitos, é importante que os entrevistadores estejam preparados e tenham uma lista de perguntas relevantes e úteis para obter as informações necessárias. As perguntas devem ser claras e concisas, evitando jargões técnicos que possam confundir o entrevistado. É importante que os entrevistados sejam escolhidos com cuidado, incluindo aqueles que são os usuários finais do software, bem como aqueles que são responsáveis por gerenciar o software.

Além disso, as entrevistas devem ser gravadas ou anotadas, para que as informações coletadas possam ser analisadas com precisão posteriormente. Essas informações serão usadas para criar um documento de requisitos, que servirá como base para o desenvolvimento do software.

Em cenários com muitos usuários, a técnica da entrevista pode ser utilizada com um grupo menor, para depois montar um questionário com os principais pontos levantados nas entrevistas, validando o entendimento com uma amostragem de usuários maior, utilizando questionários objetivos.

As entrevistas de levantamento de requisitos são uma parte crítica do processo de desenvolvimento de software e podem levar a uma série de situações diferentes. Algumas das situações que podem ocorrer durante as entrevistas de levantamento de requisitos incluem (WIEGERS; BEATTY, 2013; IIBA, 2011):

- Dificuldades de comunicação: as entrevistas podem ser realizadas com pessoas que têm diferentes níveis de conhecimento técnico e diferentes habilidades de comunicação. Isso pode levar a dificuldades na comunicação e à necessidade de o entrevistador adaptar sua abordagem para garantir que a mensagem esteja sendo transmitida e entendida corretamente.

- Requisitos incompletos ou inconsistentes: em alguns casos, os entrevistados podem não ter uma compreensão completa dos requisitos ou podem fornecer informações inconsistentes, o que pode levar a problemas na fase de desenvolvimento do software.
- Viés do entrevistado: os entrevistados podem ter suas próprias opiniões e perspectivas, o que pode levar a uma visão limitada ou distorcida dos requisitos.
- Falta de tempo ou recursos: pode haver limitações de tempo ou recursos disponíveis para realizar entrevistas detalhadas com todos os interessados no projeto, o que pode levar a requisitos omitidos ou subestimados.
- Mudanças nos requisitos: as necessidades e expectativas dos clientes e das partes interessadas podem mudar ao longo do tempo, o que pode levar a uma necessidade de alterar os requisitos originalmente definidos.
- Falta de alinhamento entre os interessados: pode haver diferenças nas expectativas e necessidades entre os diferentes interessados, o que pode levar a conflitos e dificuldades na definição de requisitos que atendam a todos os envolvidos.

Uma outra técnica muito utilizada é a observação. A observação consiste em o analista de requisitos participar, diretamente, do dia a dia da área que está pretendendo automatizar, visualizando como os processos funcionam, fazendo questionamentos, registros e observações (VAZQUEZ; SIMÕES, 2016).

A observação, que também é chamada de etnografia, um termo em referência aos estudos antropológicos de uma cultura em seu ambiente natural, pode ser uma técnica muito útil em levantamento de requisitos de software, especialmente quando o foco é entender como os usuários interagem com os processos ou com o software em seu ambiente de trabalho, em seu cotidiano (VALENTE, 2020).

Ao empregar técnicas etnográficas, os analistas de requisitos podem obter informações ricas e detalhadas sobre as necessidades dos usuários, bem como sobre as práticas e desafios que eles enfrentam em relação ao uso do software. Isso pode levar a um melhor entendimento das necessidades e expectativas dos usuários, permitindo que a solução seja desenvolvida de forma mais efetiva.

Através da observação direta dos usuários, dos processos e contextos de uso, pode ajudar a identificar problemas e oportunidades de melhoria que podem não ser evidentes em entrevistas ou questionários. Além disso, a técnica pode fornecer informações sobre o contexto cultural, social e organizacional em que o software é utilizado, que pode ser crucial para o sucesso de sua implementação.

Ao fazer um levantamento de requisitos com etnografia, ou seja, estudando as práticas e comportamentos dos usuários em seu ambiente natural, algumas situações que podem ocorrer incluem (IIBA, 2011; SOMMERVILLE, 2018):

- Descoberta de necessidades não identificadas anteriormente: a observação direta dos usuários em seu ambiente natural pode revelar necessidades que não foram previamente identificadas ou consideradas em outras abordagens de levantamento de requisitos.
- Identificação de lacunas entre o que os usuários dizem e o que eles realmente fazem: muitas vezes, os usuários podem dizer uma coisa, mas suas ações e comportamentos podem indicar algo diferente. A etnografia pode ajudar a identificar essas discrepâncias e lacunas.
- Observação de problemas de usabilidade e design: a observação direta dos usuários em seu ambiente natural pode ajudar a identificar problemas de usabilidade e design que podem não ser detectados em testes em laboratório ou entrevistas formais.
- Compreensão da cultura e contexto do usuário: a etnografia pode ajudar a entender a cultura, contexto e experiências dos usuários, o que pode ser importante para projetar soluções que sejam relevantes e significativas para eles.
- Identificação de oportunidades de melhoria: a etnografia pode ajudar a identificar oportunidades de melhoria em processos existentes e em produtos ou serviços que já estão sendo oferecidos.

Embora a etnografia seja uma técnica de coleta de requisitos muito eficaz, ela também apresenta desafios. A técnica pode ser demorada e custosa, exigindo um período prolongado de observação e interação com os usuários. Além disso, a interpretação das informações coletadas pode ser subjetiva e dependente da perspectiva do observador.

Para exemplificarmos um caso de observação, idealize que um analista de requisitos está participando de um processo de levantamento de requisitos para construir um sistema de uma loja de materiais de construção. Ele pesquisou a área de domínio do problema, entendeu basicamente como funciona uma loja deste tipo, fez algumas entrevistas e decidiu que vai passar dois dias dentro daquela loja vendo como que as coisas funcionam. Então, em um dia a dia normal, num determinando momento, chegou um caminhão com os produtos que vão ser colocados no estoque. O analista vai lá ver que o estoquista recebe os produtos e uma nota fiscal, e então começa a perguntar o que é feito com essa nota, onde que ela é armazenada, para que que ela serve e coleta todas as respostas e anota em seus arquivos. Após, um cliente chegou e foi ao caixa pagar uma parcela no crediário. O analista questiona como funciona o crediário. É uma fichinha? Tem um sistema? Como são cobrados os clientes que não pagam? Como os juros são calculados? No outro dia, ele acompanha alguém na efetuação de uma venda para entender o processo. Um pouco mais tarde, surge uma novidade: apareceu uma pessoa para trocar um produto. Não é uma coisa que não acontece toda hora, mas quando acontecer, o sistema deverá ter meios para tal. Por isso, o analista faz uma série de perguntas e guarda tudo em suas anotações.

Uma outra técnica também bem relevante, que geralmente é utilizada em uma combinação com a técnica de observação, é solicitar ativos de processos, que são documentos, formulários, planilhas para poder entender entradas, processamento e saídas de atividades executadas dentro do domínio daquele sistema.

Vamos supor que o levantamento de requisitos seja de um sistema de folha de pagamento e que um dos seus processos seja lançar as verbas salariais das pessoas além das férias, décimo terceiro salário e enviar as informações para o governo federal, para apuração do FGTS, INSS e Imposto de Renda através de arquivos. Então, solicita-se para o responsável alguns exemplos de arquivos gerados, com a maior diversidade possível, para que se possa identificar padrões, formatos, problemas e peculiaridades. Pode-se também solicitar uma ficha de cadastro de um colaborador, para poder identificar todo o tipo de informação que é relevante ser armazenada no cadastro dele.

Desta forma, é possível identificar que tipos de informações são relevantes nos processos envolvidos.

Durante um processo de análise de documentos, podem ocorrer algumas situações comuns, como (IIBA, 2011):

- Documentos desatualizados ou inconsistentes: os documentos podem estar desatualizados ou inconsistentes com a forma como os processos de negócios são atualmente executados. Isso pode levar a uma falta de clareza e precisão na compreensão dos requisitos.
- Falta de documentação: pode haver falta de documentação relevante ou ausência de informações necessárias para atender aos requisitos do projeto.
- Diferentes interpretações: diferentes pessoas podem interpretar os documentos de maneiras diferentes, o que pode levar a discrepâncias na compreensão dos requisitos.
- Conflitos entre os documentos: pode haver conflitos entre diferentes documentos, o que pode levar à confusão e incerteza sobre quais requisitos são os corretos.
- Falta de envolvimento dos stakeholders: os stakeholders relevantes podem não ter sido incluídos no processo de revisão de documentos, o que pode levar a requisitos ausentes ou mal compreendidos.
- Falha na identificação de requisitos importantes: pode haver a falha em identificar requisitos importantes durante o processo de análise de documentos, o que pode levar a problemas durante a implementação do projeto.

A grande pergunta a se fazer é: quando utilizar cada uma destas técnicas? Isso vai depender do projeto, do tipo de pessoal envolvido, de como se pretende conduzir esse trabalho. Depende, também, da quantidade e da disponibilidade de pessoas. Por exemplo, mesmo que seja um grupo pequeno que vai participar da entrevista, um questionário pode ser enviado previamente para quebrar o gelo, para que os usuários possam estar preparados para quais as perguntas serão feitas.

Deve-se tomar muito cuidado com as restrições na hora de fazer levantamento de requisitos. Tomar como verdade o resultado de um trabalho usando apenas uma das técnicas ou um grupo de usuários restrito pode ser perigoso.

O ideal é utilizar as técnicas como uma "caixa de ferramentas". Por exemplo: não está sendo possível entrevistar os usuários, eles são muito tímidos, não querem falar, ou eles não têm muito tempo. De repente, pode ser combinado com eles que as perguntas serão enviadas por escrito. Será que isso vai facilitar mais eles responderem? Depois, faz-se uma outra reunião para tirar as dúvidas.

## ATENÇÃO

Falando sobre documentos que serão utilizados para o levantamento de requisitos, é importante mencionar que o trabalho de um analista de requisitos envolve sigilo. Tanto referente a informações quanto a documentos. Além de precisar ser aderente a LGPD – Lei Geral de Proteção de Dados –, pela natureza do trabalho, lidamos com detalhes estratégicos e muitas vezes confidenciais das empresas que estamos realizando levantamento. É importante se atentar às regras de *compliance* corporativo e ter uma atuação pautada na ética e profissionalismo.

Num processo de levantamento de requisitos, o analista deve estar preparado para lidar com a resolução de conflitos. Uma das habilidades esperadas de um analista de requisitos é o controle socioemocional, além do bom relacionamento interpessoal. Na fase de levantamento de requisitos, isso precisa ser muito trabalhado. Pressman e Maxim (2016, p. 135) salientam:

Não é raro clientes e usuários pedirem mais do que é possível, dados os recursos limitados do negócio. Também é relativamente comum diferentes clientes ou usuários proporem necessidades conflitantes, argumentando que sua versão é essencial para nossas necessidades especiais. É preciso conciliar esses conflitos por meio de um processo de negociação. Devemos solicitar a clientes, usuários e outros envolvidos para que ordenem seus requisitos e discutam sua prioridade. Usando uma abordagem iterativa que priorize os requisitos, avalie seus custos e riscos e trate dos conflitos internos, os requisitos são eliminados, combinados e/ou modificados, de modo que cada parte atinja certo nível de satisfação.

O analista de requisitos precisa obter dos usuários as informações para entendimento do problema, das necessidades, dos requisitos, e, para isso, é preciso conversar, compartilhar ideias, ouvir as lamentações, lamúrias, reclamações e indignações. Quando vai ocorrer um processo de levantamento de requisitos, geralmente, aqueles usuários já estão cansados de um software anterior ou de um processo saturado, que não está dando o resultado esperado.

Além disso, mesmo nos dias atuais, existem pessoas que tem receio de um sistema novo substituir o trabalho delas. É claro que existem muitas funções que deixaram de existir ou foram transformadas por meio da tecnologia. O que precisamos

conscientizar é que a tecnologia vem para eliminar o trabalho repetitivo, o trabalho penoso. Um dos grandes exemplos é profissional de contabilidade. Se for avaliada a atividade de contabilidade antes da informática, hoje se enxerga uma evolução enorme. O contador, hoje, consegue atuar muito mais como um gestor, que utiliza sistemas para ajudar o administrador a tomar decisões baseado nos resultados.

Convencer um usuário resistente de que ele pode se beneficiar com um novo sistema pode ser um desafio, mas há algumas estratégias que um analista de requisitos pode utilizar para ajudar nesse processo (IIBA, 2011; SOMMERVILLE, 2018):

- Entender as necessidades e preocupações do usuário: é importante que o analista de requisitos tenha uma compreensão profunda das necessidades e preocupações do usuário em relação ao sistema atual. Isso permitirá que o analista possa destacar como o novo sistema pode resolver os problemas e atender às necessidades do usuário.
- Demonstrar as vantagens do novo sistema: é importante que o analista de requisitos possa demonstrar claramente os benefícios do novo sistema. Isso pode ser feito por meio de uma demonstração prática do sistema ou por meio de uma apresentação que destaque as vantagens do novo sistema em comparação com o sistema atual.
- Enfatizar a facilidade de uso do novo sistema: muitas vezes, os usuários resistem a mudanças porque têm receio de que o novo sistema seja difícil de usar. O analista de requisitos pode enfatizar a facilidade de uso do novo sistema e destacar como ele pode facilitar o trabalho do usuário.
- Oferecer treinamento e suporte: o analista de requisitos pode oferecer treinamento e suporte para ajudar os usuários a se adaptarem ao novo sistema. Isso pode incluir tutoriais em vídeo, documentação detalhada e suporte técnico para ajudar a resolver quaisquer problemas que possam surgir.
- Destacar os benefícios a longo prazo: o analista de requisitos pode enfatizar os benefícios a longo prazo do novo sistema. Isso pode incluir a economia de tempo e recursos, melhorias na eficiência do trabalho e redução de erros. Destacar esses benefícios pode ajudar a convencer o usuário de que vale a pena investir na mudança para o novo sistema.

Neste tema, vimos que técnicas de levantamento de requisitos de software são fundamentais para o desenvolvimento de sistemas de software eficazes e de alta qualidade. A coleta de requisitos é um processo crítico que envolve a identificação e documentação das necessidades e expectativas dos usuários e stakeholders, e é fundamental para o sucesso do projeto de software.

Existem diversas técnicas de levantamento de requisitos disponíveis cada uma com suas vantagens e desvantagens. É importante escolher a combinação de técnicas mais adequada para cada contexto e necessidade e aplicá-las de forma cuidadosa e rigorosa.

Um bom processo de levantamento de requisitos pode ajudar a garantir que o software desenvolvido atenda às necessidades dos usuários e stakeholders, reduzir os riscos de erros e retrabalho, e melhorar a eficiência e eficácia do processo de desenvolvimento de software. Por isso, é essencial investir tempo e recursos adequados na fase de levantamento de requisitos e priorizar a sua realização de forma plena.

# LEITURA COMPLEMENTAR

## PRODUTO MÍNIMO VIÁVEL (MVP)

Marco Tulio Valente

O conceito de MVP foi popularizado no livro Lean Startup, de Eric Ries. Por sua vez, o conceito de Lean Startup é inspirado nos princípios de Manufatura Lean, desenvolvidos por fabricantes japoneses de automóveis, como a Toyota, desde o início dos anos 1950. Já comentamos sobre Manufatura Lean no Capítulo 2, pois o processo de desenvolvimento Kanban também foi adaptado de princípios de gerenciamento de produção originados do que ficou conhecido depois como Manufatura Lean. Um dos princípios de Manufatura Lean recomenda eliminar desperdícios em uma linha de montagem ou cadeia de suprimentos. No caso de uma empresa de desenvolvimento de software, o maior desperdício que pode existir é passar anos levantando requisitos e implementando um sistema que depois não vai ser usado, pois ele resolve um problema que não interessa mais a seus usuários. Portanto, se é para um sistema falhar — por não ter sucesso, usuários ou mercado — é melhor falhar rapidamente, pois o desperdício de recursos será menor.

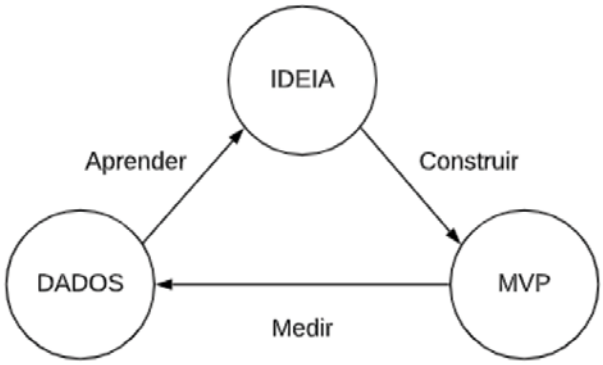
Sistemas de software que não atraem interesse podem ser produzidos por qualquer empresa. No entanto, eles são mais comuns em startups, pois por definição elas são empresas que operam em ambientes de grande incerteza. No entanto, Eric Ries também lembra que a definição de startup não se restringe a uma empresa formada por dois universitários que desenvolvem em uma garagem um novo produto de sucesso instantâneo. Segundo ele, qualquer pessoa que está criando um novo produto ou negócio sob condições de extrema incerteza é um empreendedor, quer saiba ou não, e quer trabalhe em uma entidade governamental, em uma empresa apoiada por capital de risco, em uma organização sem fins lucrativos ou em uma empresa com investidores financeiros decididamente voltada para o lucro.

Então, para deixar claro o nosso cenário, suponha que pretendemos criar um sistema novo, mas não temos certeza de que ele terá usuários e fará sucesso. Como comentado, não vale a pena passar um ou dois anos levantando os requisitos desse sistema, para então concluir que ele será um fracasso. Por outro lado, não faz muito sentido também realizar pesquisas de mercado, para aferir a receptividade do sistema antes de implementá-lo. Como ele é um sistema novo, com requisitos diferentes de quaisquer sistemas existentes, os resultados de uma pesquisa de mercado podem não ser confiáveis.

Uma solução consiste em implementar um sistema simples, com um conjunto de requisitos mínimos, mas que sejam suficientes para testar a viabilidade de continuar investindo no seu desenvolvimento. Em Lean Startup, esse primeiro sistema é chamado de Produto Mínimo Viável (MVP). Costuma-se dizer também que o objetivo de um MVP é testar uma hipótese de negócio.

Lean startup propõe um método sistemático e científico para construção e validação de MVPs. Esse método consiste em um ciclo com três passos: construir, medir e aprender (veja próxima figura). No primeiro passo (construir), tem-se uma ideia de produto e então implementa-se um MVP para testá-la. No segundo passo (medir), o MVP é disponibilizado para uso por clientes reais com o intuito de coletar dados sobre a sua viabilidade. No terceiro passo (aprender), as métricas coletadas são analisadas e geram o que se denomina de aprendizado validado (validated learning).

Figura – Método Lean Startup para Validação de MVPs




O aprendizado obtido com um MVP pode resultar em três decisões:

- Pode-se concluir que ainda são necessários mais testes com o MVP, possivelmente alterando seu conjunto de requisitos, sua interface com os usuários ou o mercado alvo. Logo, repete-se o ciclo, voltando para o passo construir.
- Pode-se concluir que o teste foi bem-sucedido e, portanto, achou-se um mercado para o sistema (um market fit). Neste caso, é hora de investir mais recursos para implementar um sistema com um conjunto mais robusto e completo de funcionalidades.
- Por fim, pode-se concluir que o MVP falhou, após várias tentativas. Então, restam duas alternativas: (1) perecer, isto é, desistir do empreendimento, principalmente se não existirem mais recursos financeiros para mantê-lo vivo; ou (2) realizar um pivô, isto é, abandonar a visão original e tentar um novo MVP, com novos requisitos e para um novo mercado, mas sem esquecer o que se aprendeu com o MVP anterior.

Ao tomar as decisões, um risco é usar apenas métricas de vaidade (*vanity metrics*). Essas são métricas superficiais que fazem bem para o ego dos desenvolvedores e gerentes de produto, mas que não ajudam a entender e aprimorar uma estratégia

de mercado. O exemplo clássico é o número de *pageviews* em um site de comércio eletrônico. Pode fazer muito bem dizer que o site atrai milhões de clientes por mês, mas somente isso não vai ajudar a pagar as contas do empreendimento. Por outro lado, métricas que ajudam a tomar decisões sobre o futuro de um MVP são chamadas de métricas acionáveis (*actionable metrics*). No caso de um sistema de comércio eletrônico, essas métricas incluiriam o percentual de visitantes que fecham compras, o valor de cada ordem de compra, o número de itens comprados, o custo de captação de novos clientes etc. Ao monitorar essas métricas, pode-se concluir, por exemplo, que a maioria dos clientes compra apenas um item ao fechar uma compra. Como resultado concreto — ou acionável — pode-se decidir incorporar um sistema de recomendação ao site ou, então, investigar o uso de um sistema de recomendação mais eficiente. Tais sistemas, dada uma compra em andamento, são capazes de sugerir novos itens para serem comprados. Assim, eles têm o potencial de incrementar o número de itens comprados em uma mesma transação.

Para avaliar MVPs que incluem vendas de produtos ou serviços, costuma-se usar métricas de funil (*funnel metrics*), que capturam o nível de interação dos usuários com um sistema. Um funil pode incluir as seguintes métricas:

- Aquisição: número de clientes que visitaram o seu sistema.
- Ativação: número de clientes que criaram uma conta no sistema.
- Retenção: clientes que retornaram ao sistema, após criarem uma conta.
- Receita: número de clientes que fizeram uma compra.
- Recomendação: clientes que recomendaram o sistema para terceiros.

**Exemplos de MVP**

Um MVP não precisa ser um software real, implementado em uma linguagem de programação, com bancos de dados, integração com outros sistemas etc. Dois exemplos de MVP que não são sistemas são frequentemente mencionados nos artigos sobre Lean Startup.

O primeiro é o caso da Zappos, uma das primeiras empresas a tentar vender sapatos pela Internet nos Estados Unidos. Em 1999, para testar de forma pioneira a viabilidade de uma loja de sapatos virtual, o fundador da empresa concebeu um MVP simples e original. Ele visitou algumas lojas de sapatos de sua cidade, fotografou diversos pares de sapato e criou uma página Web bastante simples, por meio da qual os clientes poderiam selecionar os sapatos que desejassem comprar. Porém, todo o processamento era feito de forma manual, incluindo a comunicação com a empresa de cartões de crédito, a compra dos sapatos nas lojas da cidade e a remessa para os clientes. Não existia nenhum sistema para automatizar essas tarefas. No entanto, com esse MVP baseado em tarefas manuais, o dono da Zappos conseguiu validar de forma rápida e barata a sua hipótese inicial, isto é, de que havia mercado para venda de sapatos pela Internet. Anos mais tarde, a Zappos foi adquirida pela Amazon, por mais de um bilhão de dólares.

Um segundo exemplo de MVP que não envolveu a disponibilização de um software real para os usuários vem do Dropbox (o sistema de compartilhamento e armazenamento de arquivos na nuvem). Para receber feedback sobre o produto que estavam desenvolvendo, um dos fundadores da empresa gravou um vídeo simples, quase amador, demonstrando em 3 minutos as principais funcionalidades e vantagens do sistema que estavam desenvolvendo. O vídeo viralizou e contribuiu para aumentar a lista de usuários interessados em realizar um teste do sistema (de 5 mil para 75 mil usuários). Outro fato interessante é que os arquivos usados no vídeo tinham nomes engraçados e que faziam referência a personagens de histórias em quadrinhos. O objetivo era chamar a atenção de adotantes iniciais (*early adopters*), que são aquelas pessoas aficionadas por novas tecnologias e que se dispõem a serem as primeiras a testar e comprar novos produtos. A hipótese que se queria validar com o MVP em forma de vídeo é que havia usuários interessados em instalar um sistema de sincronização e backup de arquivos. Essa hipótese se revelou verdadeira pela atração de um grande número de adotantes iniciais dispostos a fazer um teste beta do Dropbox.

No entanto, MVPs também podem ser implementados na forma de sistemas de software reais, embora mínimos. Por exemplo, no início de 2018, nosso grupo de pesquisa na UFMG iniciou o projeto de um sistema para catalogar a produção científica brasileira em Ciência da Computação. A primeira decisão foi construir um MVP, cobrindo apenas artigos em cerca de 15 conferências da área de Engenharia de Software. Nessa primeira versão, o código implementado em Python tinha menos de 200 linhas. Os gráficos mostrados pelo sistema, por exemplo, eram planilhas do Google Spreadsheets embutidas em páginas HTML. Esse sistema — inicialmente chamado CoreBR — foi divulgado e promovido em uma lista de e-mails da qual participam os professores brasileiros de Engenharia de Software. Como o sistema atraiu um bom interesse, medido por meio de métricas como duração das sessões de uso, decidimos investir mais tempo na sua construção. Primeiro, seu nome foi alterado para CSIndexbr (link). Depois, expandimos gradativamente a cobertura para mais 20 áreas de pesquisa em Ciência da Computação e quase duas centenas de conferências. Passamos a cobrir também artigos publicados em mais de 170 periódicos. O número de professores com artigos indexados aumentou de menos de 100 para mais de 900 professores. A interface do usuário deixou de ser um conjunto de planilhas e passou a ser um conjunto de gráficos implementados em JavaScript.

**Perguntas Frequentes**

Para finalizar, vamos responder algumas perguntas sobre MVPs.

**Apenas startups devem usar MVPs?** Definitivamente não. Como tentamos discutir nesta seção, MVPs são um mecanismo para lidar com incerteza. Isto é, quando não sabemos se os usuários vão gostar e usar um determinado produto. No contexto de Engenharia de Software, esse produto é um software. Claro que startups, por definição, são empresas que trabalham em mercados de extrema incerteza. Porém, incerteza

e riscos também podem caracterizar software desenvolvido por diversos tipos de organização, privadas ou públicas; pequenas, médias ou grandes; e dos mais diversos setores.

**Quando não vale a pena usar MVPs?** De certo modo, essa pergunta foi respondida na questão anterior. Quando o mercado de um produto de software é estável e conhecido, não há necessidade de validar hipóteses de negócio e, portanto, de construir MVPs. Em sistemas de missão crítica, também não se cogita a construção de MVPs. Por exemplo, está fora de cogitação construir um MVP para um software de monitoramento de pacientes de UTIs.

**Qual a diferença entre MVPs e prototipação?** Prototipação é uma técnica conhecida em Engenharia de Software para elicitação e validação de requisitos. A diferença entre protótipos e MVPs está nas três letras da sigla, isto é, tanto no M, como no V e no P. Primeiro, protótipos não são necessariamente sistemas mínimos. Por exemplo, eles podem incluir toda a interface de um sistema, com milhares de funcionalidades. Segundo, protótipos não são necessariamente implementados para testar a viabilidade de um sistema junto aos seus usuários finais. Por exemplo, eles podem ser construídos para demonstrar o sistema apenas para os executivos de uma empresa contratante. Por isso mesmo, eles também não são produtos.

**Um MVP é um produto de baixa qualidade?** Essa pergunta é mais complexa de ser respondida. Porém, é verdade que um MVP deve ter apenas a qualidade mínima necessária para avaliar um conjunto de hipóteses de negócio. Por exemplo, o código de um MVP não precisa ser de fácil manutenção e usar os mais modernos padrões de design e frameworks de desenvolvimento, pois pode ser que o produto se mostre inviável e seja descartado. Na verdade, em um MVP, qualquer nível de qualidade a mais do que o necessário para iniciar o laço construir-medir-aprender é considerado desperdício. Por outro lado, é importante que a qualidade de um MVP não seja tão ruim a ponto de impactar negativamente a experiência do usuário. Por exemplo, um MVP hospedado em um servidor Web com problemas de disponibilidade pode dar origem a resultados chamados de falsos negativos. Eles ocorrem quando a hipótese de negócio é falsamente invalidada. No nosso exemplo, o motivo do insucesso não estaria no MVP, mas sim no fato de os usuários não conseguirem acessar o sistema, pois o servidor frequentemente estava fora do ar.

### Construindo o Primeiro MVP

Lean startup não define como construir o primeiro MVP de um sistema. Em alguns casos isso não é um problema, pois os proponentes do MVP têm uma ideia precisa de suas funcionalidades e requisitos. Então, eles já conseguem implementar o primeiro MVP e, assim, iniciar o ciclo construir-medir-aprender. Por outro lado, em certos casos, mesmo a ideia do sistema pode não estar clara. Nesses casos, recomenda-se construir um protótipo antes de implementar o primeiro MVP.

Design Sprint é um método proposto por Jake Knapp, John Zeratsky e Braden Kowitz para testar e validar novos produtos por meio de protótipos, não necessariamente de software (link). As principais características de um design sprint — não confundir com um sprint, de Scrum — são as seguintes:

- Time-box. Um design sprint tem a duração de cinco dias, começando na segunda-feira e terminando na sexta-feira O objetivo é descobrir uma primeira solução para um problema rapidamente.
- Equipes pequenas e multidisciplinares. Um design sprint deve reunir uma equipe multidisciplinar de sete pessoas. Ao definir esse tamanho, o objetivo é fomentar discussões — por isso, a equipe não pode ser muito pequena. Porém, procura-se evitar debates intermináveis — por isso, a equipe não pode também ser muito grande. Da equipe, devem participar representantes de todas as áreas envolvidas com o sistema que se pretende prototipar, incluindo pessoas de marketing, vendas, logística etc. Por último, mas não menos importante, a equipe deve incluir um tomador de decisões, que pode ser, por exemplo, o próprio dono da empresa.
- Objetivos e regras claras. Os três primeiros dias do design sprint têm como objetivo convergir, depois divergir e, então, convergir novamente. Isto é, no primeiro dia, entende-se e delimita-se o problema que se pretende resolver. O objetivo é garantir que, nos dias seguintes, a equipe estará focada em resolver o mesmo problema (convergência). No segundo dia, possíveis alternativas de solução são propostas, de forma livre (divergência). No terceiro dia, escolhe-se uma solução vencedora, dentre as possíveis alternativas (convergência). Nessa escolha, a última palavra cabe ao tomador de decisões, isto é, um design sprint não é um processo puramente democrático. No quarto dia, implementa-se um protótipo, que pode ser simplesmente um conjunto de páginas HTML estáticas, sem qualquer código ou funcionalidade. No último dia, testa-se o protótipo com cinco clientes reais, com cada um deles usando o sistema em sessões individuais.

Antes de concluir, é importante mencionar que design sprint não é voltado apenas para definição de um protótipo de MVP. A técnica pode ser usada para propor uma solução para qualquer problema. Por exemplo, pode-se organizar um design sprint para reformular a interface de um sistema, já em produção, mas que está apresentando uma alta taxa de abandono.

Fonte: VALENTE, M. T. **Engenharia de Software Moderna**: Princípios e Práticas para Desenvolvimento de Software com Produtividade. 2020. Disponível em: https://engsoftmoderna.info. Acesso em: 18 dez. 2022.

# RESUMO DO TÓPICO 3

**Neste tópico, você aprendeu:**

- O quanto o levantamento de requisitos é essencial para o processo de desenvolvimento de software.

- A técnica de mapeamento de processos de negócio BPMN, que permite que as organizações conheçam e documentem seus processos e possam automatizá-los por meio de sistemas de uma forma mais assertiva.

- O formato criativo do design *thinking* pode ser utilizado para desenvolvimento de software, e é muito bem aplicado neste contexto para permitir a experimentação, a criação e validação de MVPs.

- Técnicas diversas de levantamento de requisitos combinadas podem trazer resultados excelentes para a definição do escopo de um projeto de software, considerando as peculiaridades de cada projeto e o tipo e engajamento das partes interessadas envolvidas.

# AUTOATIVIDADE

1 O BPMN é uma linguagem visual que permite que as organizações documentem seus processos de negócio. Sobre os principais elementos da notação BPMN, assinale a alternativa CORRETA:

a) ( ) Atividades, gateways, eventos, conectores e raias.
b) ( ) Processos, atividades, retornos, conexões.
c) ( ) Atividades, eventos, conectores, disparos e paralisações.
d) ( ) Processos, atividades, retornos, conexões e eventos.

2 O *design thinking* tem como objetivo apoiar o processo de resolução de problemas complexos utilizando formas criativas de se trabalhar com estes desafios. É sugerida uma série de etapas para se trabalhar a construção de um produto utilizando a metodologia. Sobre a sequência em que ocorre essa série de etapas, assinale a alternativa CORRETA:

a) ( ) Imersão; Análise e Síntese; Ideação; Prototipação; Implementação Final.
b) ( ) Ideação; Análise e Síntese; Imersão; Prototipação; Implementação Final.
c) ( ) Imersão; Ideação; Análise e Síntese; Prototipação; Implementação Final.
d) ( ) Análise e Síntese; Imersão; Ideação; Prototipação; Implementação Final.

3 Entre as técnicas de levantamento de requisitos, pode-se destacar a compreensão do domínio do problema. Sobre em que consiste a compreensão do domínio do problema, assinale a alternativa CORRETA:

a) ( ) Observar as partes interessadas executando as tarefas do dia a dia.
b) ( ) Estudar sobre o contexto de negócio que envolve o sistema a ser desenvolvido.
c) ( ) Procurar meios de resolver o problema usando técnicas avançadas.
d) ( ) Realizar uma reunião de brainstorm com o intuito de obter ideias para resolução do problema.

4 Iniciar um processo de levantamento de requisitos, uma entrevista com um usuário, pode ser difícil para a maioria dos engenheiros de requisitos iniciantes. Além de realizar uma compreensão do domínio do problema, que perguntas podem ser realizadas para iniciar este levantamento, independentemente do tipo de sistema a ser construído?

5 Num contexto de desenvolvimento de um software para o governo federal, o analista de requisitos de deparou com um número de partes interessadas muito alto. Para executar o projeto, sem prejuízo o entendimento das necessidades, como pode-se resolver este problema? Justifique.

# REFERÊNCIAS


BECK, K. *et al*. **Manifesto Ágil.** 2001. Disponível em: http://paginapessoal.utfpr.edu.br/frufrek/pos-web/p/arquivos/0_manifesto_agil.pdf. Acesso em: 20 dez. 2022.

BIESDORF, S.; COURT, D.; WILLMOTT, P. **Big Data**: O Próximo Desafio para o Crescimento e Inovação das Empresas. Rio de Janeiro: Elsevier, 2013.

BROOKS JR., F. P. No Silver Bullet Essence and Accidents of Software Engineering. **Computer**, [S.L.], v. 20, n. 4, p. 10-19, abr. 1986.

BROWN, T. **Design Thinking**: uma metodologia poderosa para decretar fim as velhas ideias. Rio de Janeiro: Alta Books, 2018.

CAMPOS, A. L. N. **Modelagem de Processos com BPMN**. 2. ed. Rio de Janeiro: Brasport, 2014.

DRUCKER, P. **Managing for Business Effectiveness**. 1963. Disponível em: https://hbr.org/1963/05/managing-for-business-effectiveness. Acesso em: 18 dez. 2022.

IEEE. ***Guide to the Software Engineering Body of Knowledge: SWEBOK® v. 3.0***. IEEE Computer Society, 2014.

IIBA. **Um guia para o corpo de conhecimento de análise de negócios: Guia BABOK®**. *versão 2.0*. Toronto: theiiba.org, 2011.

KERR, E. S. **Gerenciamento de Requisitos**. São Paulo: Pearson, 2015

PFLEEGER, S. L. **Engenharia de software**: teoria e prática. 2 ed. São Paulo: Prentice Hall, 2004

PMI. **Um guia do conhecimento em gerenciamento de projetos**. Guia PMBOK® 6a. ed. Newtown Square, PA: Project Management Institute, 2017

PMI. **Pulse of the Profession 2018**: success in disruptive times. 2018. Disponível em: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf. Acesso em: 21 dez. 2022.

POHL, K; RUPP, C. **Requirements Engineering Fundamentals**: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level. Santa Barbara, CA: Rocky Nook Inc., 2011.

PRESSMAN, R.; MAXIM, B. R. **Engenharia de Software**: uma abordagem profissional. 8. ed. Porto Alegre: Amgh, 2016.

MARCO, T. de; LISTER, T. **Peopleware**: productive projects and teams. 2. ed. Nova Iorque: Dorset House Publishing Co, 1999.

MOHANANI, R. *et al.* Cognitive Biases in Software Engineering: a systematic mapping study. **Ieee Transactions On Software Engineering**, v. 46, n. 12, p. 1318-1339, 1 dez. 2020. Disponível em: http://dx.doi.org/10.1109/tse.2018.2877759. Acesso em: 28 abr. 2023.

MELLO, C. de M.; ALMEIDA NETO, J. R. M. de; PETRILLO, R. P. **Para compreender o Design Thinking**. Rio de Janeiro: Processo, 2021.

OMG. **Business Process Model and Notation (BPMN)**. 2011. Disponível em: https://www.omg.org/spec/BPMN/2.0/PDF. Acesso em: 20 dez. 2022.

SCHMIDT, R. *et al*. **Identificando os riscos do projeto de software**: um estudo Delphi internacional. Journal of Management Information Systems. 2021.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. **Administração da Produção**. 3. ed. São Paulo: Atlas, 2009.

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro**: guia geral MPS de software. 2021. Disponível em: https://softex.br/download/guia-geral-de-software-2021/. Acesso em: 20 dez. 2022.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018

WIEGERS, K. E.; BEATTY, J. **Software Requirements**. 3rd Edition. Redmond: Microsoft Press, 2013.

VALENTE, M. T. **Engenharia de Software Moderna**: Princípios e Práticas para Desenvolvimento de Software com Produtividade. 2020. Disponível em: https://engsoftmoderna.info. Acesso em: 18 dez. 2022.

VAZQUEZ, C. E.; SIMÕES, G. S. **Engenharia de Requisitos**: software orientado ao negócio. Rio de Janeiro: Brasport, 2016.

# ESPECIFICAÇÃO DE REQUISITOS – FERRAMENTAS E TÉCNICAS

## OBJETIVOS DE APRENDIZAGEM

**A partir do estudo desta unidade, você deverá ser capaz de**:

- compreender a necessidade de classificar os requisitos e descrevê-los da forma adequada;
- construir uma documentação de especificação de requisitos completa e detalhada;
- conduzir processos de escrita de histórias de uso de forma colaborativa junto às partes interessadas;
- conduzir a criação de diagramas de caso de uso e de atividade para melhor comunicar os objetivos das funcionalidades de um sistema;

## PLANO DE ESTUDOS

A cada tópico desta unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TEMA DE APRENDIZAGEM 1 – CLASSIFICAÇÃO DE REQUISITOS

TEMA DE APRENDIZAGEM 2 – DOCUMENTAÇÃO DE ESPECIFICAÇÃO DE REQUISITOS

TEMA DE APRENDIZAGEM 3 – UML – UNIFIED MODEL LANGUAGE

## CHAMADA

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

# CONFIRA A **TRILHA** DA UNIDADE 2!

Acesse o
QR Code abaixo:

# CLASSIFICAÇÃO DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 1, abordaremos a classificação de requisitos de software que é uma etapa crucial no desenvolvimento de um software. É aí que se define e prioriza as funcionalidades e características que o software precisa ter para atender às expectativas do usuário. A classificação dos requisitos é uma tarefa importante porque ajuda a garantir que o software seja desenvolvido de acordo com as necessidades do cliente e dos usuários finais. Além disso, ela permite uma gestão mais eficiente do projeto, pois permite que o time de desenvolvimento tenha uma visão clara dos requisitos e possa trabalhar com eficiência na sua implementação.

Os requisitos de software podem ser classificados de diversas maneiras, dependendo do contexto do projeto e do objetivo que se quer alcançar. Algumas das classificações mais comuns são por tipo (funcional, não funcional), por origem (usuário, sistema), por criticidade (obrigatório, desejável etc.) ou por fase do ciclo de vida (requisitos de negócio, requisitos de sistema). Independentemente da classificação escolhida, o importante é que ela permita uma compreensão clara dos requisitos e uma gestão eficiente do projeto.

Na prática, a classificação de requisitos é realizada por meio de uma análise cuidadosa dos requisitos coletados. É importante envolver todas as partes interessadas no processo, incluindo o cliente, os usuários finais, o time de desenvolvimento e outras partes interessadas, para garantir que todas as perspectivas sejam consideradas. Além disso, é importante documentar a classificação dos requisitos de forma clara e objetiva, para que ela possa ser revisada e atualizada ao longo do projeto, conforme as necessidades mudem.

## 2 CLASSIFICAÇÃO DE REQUISITOS

Em sua obra, Sommerville (2018, p. 85) afirma que "requisitos de um sistema são as descrições dos serviços que o sistema deve prestar e as restrições a sua operação". Sommerville (2018) apresenta a necessidade de distinguir os requisitos em duas categorias: requisitos de usuário, que são requisitos abstratos declarados em linguagem natural e alto nível e requisitos de sistema, que devem conter uma descrição detalhada do que o sistema deve fazer.

Para que esta diferença fique mais clara, veja o exemplo de requisitos relacionados a uma lanchonete:

**Requisito de usuário**: O sistema deve permitir que os clientes possam fazer pedidos de lanches, bebidas e sobremesas em uma caixa de lanchonete e receber seus pedidos com rapidez e eficiência.

**Requisito de sistema:** O sistema de caixa de lanchonete deve permitir a entrada de pedidos de forma eficiente e precisa, integrando-se com o sistema de cozinha para agilizar a preparação dos pedidos. O sistema deve permitir a impressão de pedidos em papel ou exibição em monitores, de acordo com a preferência da lanchonete. Além disso, o sistema deve permitir o cálculo automático do valor total do pedido, incluindo impostos e eventuais descontos, e a impressão de recibos para os clientes.

Sommerville (2018) afirma que é necessário descrever requisitos em diferentes níveis de aprofundamento de acordo com o tipo de vínculo e interesse que leitores destes requisitos tem do sistema. Ele defende que requisitos de usuário estão mais voltados para gerentes do cliente, usuários finais do sistema, engenheiros do cliente, gerentes da equipe contratada e arquitetos do sistema. Já requisitos de sistema estão vinculados a usuários finais do sistema, engenheiros do cliente, arquitetos do sistema e desenvolvedores de

Para Pressman e Maxim (2016), os requisitos de software são divididos em duas categorias principais, os funcionais e não funcionais. Os requisitos funcionais têm ligação com o que o software deve fazer. Os não funcionais estão vinculados a características de qualidade e arquiteturais do software.

A tradução dos resultados obtidos utilizando diversas técnicas da etapa de levantamento de requisitos deve construir uma relação de requisitos funcionais e não funcionais, que permite que os desenvolvedores compreendam claramente o que é esperado do sistema, e isso ajuda a garantir que ele seja desenvolvido de acordo com as expectativas das partes interessadas. Além disso, classificar os requisitos em categorias ajuda a garantir que todos os requisitos importantes sejam considerados e atendidos, e que não haja confusão quanto às expectativas do sistema.

## 2.1 REQUISITOS FUNCIONAIS

Para Sommerville (2018, p. 88), os requisitos funcionais "são declarações dos serviços que o sistema deve fornecer, do modo como o sistema deve reagir a determinadas entradas e de como deve se comportar em determinadas situações". Isso inclui as necessidades específicas dos usuários para resolver os problemas de negócio que ele possui. Em sua obra, Sommerville (2018) ainda destaca que além do estabelecimento de serviços que o sistema deve fornecer, um requisito funcional deve apresentar o comportamento do sistema em situações específicas, bem como pode declaradamente definir o que o sistema não deve fazer.

Os requisitos funcionais também devem ser rastreáveis e testáveis, para garantir que o sistema seja desenvolvido de acordo com as especificações e que as funcionalidades estejam operando corretamente. Segundo IEEE (2017), a rastreabilidade dos requisitos é fundamental para o gerenciamento de mudanças e para a garantia da qualidade do software.

A rastreabilidade de requisitos consiste na capacidade de identificar e documentar as relações entre diferentes requisitos e entre requisitos e outros artefatos de um projeto de software, como testes, casos de uso e código-fonte. De acordo com IEEE (2017), a rastreabilidade de requisitos é definida como "a capacidade de rastrear a origem, o destino e o status de cada requisito e suas relações com outros requisitos ao longo do ciclo de vida do software"

Já a definição de requisitos testáveis é importante porque ajuda a garantir que o sistema atenda às expectativas do cliente e que as funcionalidades estejam operando corretamente. Segundo Pressman e Maxim (2016), os requisitos testáveis ajudam a garantir que o sistema faça o que foi especificado e que o software produzido seja de alta qualidade.

Para exemplificar o que são requisitos funcionais, vamos imaginar o cenário de uma família pretendo construir uma casa que atenda todas as suas necessidades. Todas as partes interessadas foram ouvidas, o pai, a mãe e os filhos, a lista de requisitos funcionais para a construção da casa ficou da seguinte forma:

- A residência deve possuir uma cozinha com duas janelas de 1.20m x 1.20m.
- A residência deve possuir no mínimo 3 dormitórios.
- A residência deve possuir 1 banheiro vinculado a cada dormitório.
- A residência deve possuir grades de proteção em todas as portas e janelas.
- Em todas as torneiras de água deve estar disponível água quente e fria.
- A residência deve possuir uma sala de estar.
- A residência deve possuir uma garagem coberta para dois carros.

Veja que na lista especificamos os requisitos de acordo com as necessidades repassadas pelas partes interessadas (neste caso a família), até um determinado nível de detalhe. Em alguns casos, temos mais especificidades, características, em outros as definições ficam mais em aberto, por exemplo, no primeiro requisito, determina-se o tamanho das janelas, mas não fala de que material a janela deve ser (exemplo: madeira, ferro, alumínio).

No segundo requisito, tem-se que deve ter no mínimo três dormitórios, mas não menciona uma metragem mínima, nem se precisa ter janelas. Parece um pouco exagerado? Todo dormitório precisa ter uma janela certo? Então, por que não detalhar? Observe também que fica difícil referenciar cada requisito mencionando apenas primeiro ou segundo.

Considerar que existam obviedades, um senso comum, em processos de engenharia de requisitos, é um erro. Wiegers e Beatty (2013) enfatizam a importância de documentar todos os requisitos, mesmo aqueles que parecem óbvios, para garantir que eles sejam compreendidos por todas as partes envolvidas no projeto. Embora possa parecer óbvio, não assuma que todos os outros compartilham o seu conhecimento. Documente-o de qualquer maneira, para que possa ser revisado e confirmado ou discutido.

Para ter um detalhamento melhor, e uma forma melhor de identificação, será apresentada a seguir uma reescrita dos requisitos da residência.

- RF01. A residência deve possuir uma cozinha com duas janelas de alumínio branco com o tamanho mínimo de 1.20m x 1.20m.
- RF02. A residência deve possuir no mínimo 3 dormitórios, com no mínimo 12 metros quadrados, uma porta de acesso de madeira com 0,80m metros de largura por 2,10m de altura e uma janela de alumínio branco com o tamanho mínimo de 1.20m x 1.20m.
- RF03. A residência deve possuir 1 banheiro vinculado a cada dormitório com um balcão de pia, um vaso sanitário e uma área de banho, totalizando no mínimo 3m2.
- RF04. A residência deve possuir grades de proteção de ferro em todas as portas e janelas.
- RF05. Em todas as torneiras de água deve estar disponível água quente e fria.
- RF06. A residência deve possuir uma sala de estar com no mínimo 12m2.
- RF07. A residência deve possuir uma garagem coberta para dois carros lado a lado, com o tamanho mínimo de 25m2.

Veja que nesta nova versão da lista de requisitos da residência especificamos mais detalhes em cada requisito, permitindo um aprofundamento melhor das necessidades. Uma lista de requisitos inicial é preparada baseada nos trabalhos de levantamento de requisitos, e ela deve ser refinada e revisada continuadamente, com o trabalho dos engenheiros de requisitos e das partes interessadas envolvidas.

Um outro aspecto que foi agregado a esta nova lista foi a identificação, cada requisito recebeu uma identificação única, a sigla de requisito funcional (RF) e um numeral. Desta forma, ao referenciar um requisito, é possível identificá-lo de forma fácil, seja numa reunião, seja num e-mail, ou em um documento de alteração de escopo. A numeração nos requisitos pode também facilitar, já que é mais fácil classificá-los por ordem de importância. O engenheiro de requisitos poderá facilmente descrever, por exemplo: “foi necessário alterar o RF04 para incluir os detalhes discutidos na última reunião”. Desta forma, pode ser rapidamente identificado por todos.

Agora que entendemos que os requisitos funcionais estão ligados essencialmente à funcionalidade da residência, veremos requisitos funcionais de um sistema. São exemplos de requisitos funcionais de um sistema de controle de veículos de uma empresa:

- RF01 - O sistema deve permitir o cadastro de veículos, com no mínimo as seguintes informações: marca, modelo, ano de fabricação/ano modelo, placa e cor.
- RF02 - O sistema deve permitir o registro do abastecimento de cada um dos veículos, contendo funcionário responsável, data do abastecimento, combustível, quantidade de litros e valor total.
- RF03 - O sistema deve calcular a média de consumo para cada veículo especificamente e para cada modelo.
- RF04 - O usuário poderá incluir os acessórios pertencentes a cada veículo.
- RF05 - O sistema deve emitir um relatório com os abastecimentos, permitindo realizar filtros por funcionário responsável, placa e data do abastecimento.
- RF06 - O sistema deve permitir que o usuário cadastre os motoristas responsáveis por cada veículo, incluindo informações como nome, CPF e CNH.
- RF07 - O sistema deve permitir o registro de manutenções realizadas em cada veículo, incluindo a data, o tipo de manutenção e o valor gasto.
- RF08 - O sistema deve permitir a criação de alertas para a realização de manutenções preventivas em cada veículo, baseado na quilometragem percorrida.
- RF09 - O sistema deve permitir que o usuário cadastre fornecedores de combustível, incluindo informações como nome, CNPJ e endereço.
- RF10 - O sistema deve permitir que o usuário visualize um histórico completo de manutenções de cada veículo, ordenado por data.

Os requisitos funcionais desempenham um papel crucial no processo de desenvolvimento de software. Eles descrevem as funcionalidades esperadas do software e são a base para a implementação e teste do software. A escrita de requisitos funcionais claros, precisos e detalhados é uma prática importante para garantir que o projeto tenha sucesso e atenda às expectativas dos stakeholders. Ao seguir as boas práticas recomendadas, como envolver as partes interessadas, manter os requisitos claros e precisos, ser detalhado, priorizar os requisitos e garantir que os requisitos sejam testáveis, você pode aumentar as chances de sucesso do seu projeto de software.

Para Pressman e Maxim (2016), "a qualidade dos requisitos de software é um fator crítico para o sucesso do projeto. Requisitos mal definidos, incompletos ou inconsistentes são uma das principais causas de fracasso de projetos de software".

Imagine uma situação em que uma empresa decidiu desenvolver um software de gerenciamento de projetos sem escrever os requisitos funcionais. Durante o processo de desenvolvimento, o time de desenvolvimento decidiu por sua conta e risco sobre as funcionalidades que o software deveria ter e começou a implementá-las.

No entanto, quando o software foi entregue às partes interessadas para testes, eles descobriram que ele não atendia as suas expectativas e necessidades. Algumas funcionalidades importantes estavam faltando, enquanto outras funcionalidades eram desnecessárias e desperdiçavam recursos. Além disso, alguns recursos implementados estavam implementados de forma incorreta e precisavam ser corrigidos.

Como resultado, a empresa gastou muito tempo e recursos adicionais corrigindo os erros e implementando as funcionalidades faltantes, e ainda assim, o software não atendeu completamente às necessidades da organização. A falta de requisitos funcionais claros e precisos leva a erros caros e a uma entrega final de um software que "não serve".

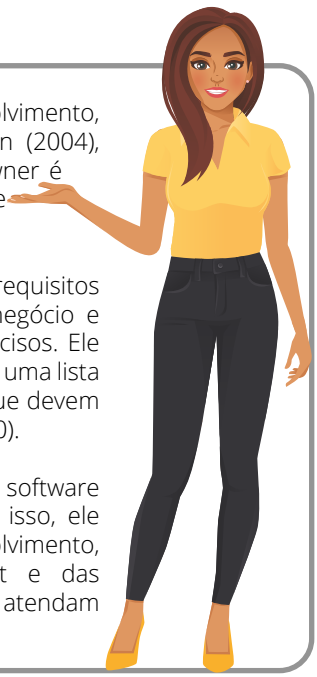
Além disso, é importante revisar e atualizar os requisitos funcionais à medida que o projeto evolui para garantir que eles ainda sejam relevantes e atendam às necessidades do projeto. Envolver as partes interessadas e outros membros do time de desenvolvimento na revisão e atualização de requisitos vai garantir que todas as perspectivas importantes sejam consideradas.

## INTERESSANTE

Dentro das equipes que utilizam metodologias ágeis de desenvolvimento, um dos papéis existentes é o de Product Owner. Para Cohn (2004), ele representa o cliente ou o dono do produto. O Product Owner é responsável por definir e priorizar os requisitos do software de acordo com as necessidades do negócio e dos usuários finais.

O Product Owner tem um papel fundamental na definição dos requisitos de software, pois é ele quem entende as necessidades do negócio e dos usuários finais e os transforma em requisitos claros e concisos. Ele também é responsável por gerenciar o backlog do produto, que é uma lista ordenada de todas as funcionalidades, requisitos e melhorias que devem ser implementados no software (SUTHERLAND; SCHWABER, 2020).

Além disso, o Product Owner é responsável por garantir que o software atenda às expectativas do cliente e dos usuários finais. Para isso, ele deve estar constantemente envolvido no processo de desenvolvimento, participando das reuniões diárias, das revisões de sprint e das retrospectivas para garantir que as funcionalidades desenvolvidas atendam às necessidades do negócio e dos usuários finais.



## 2.2 REQUISITOS NÃO FUNCIONAIS

O IIBA (2011) define que os requisitos não funcionais estão relacionados ao ambiente no qual o software deve permanecer operante. Para o IEEE (2014), os requisitos não funcionais atuam para restringir as soluções. Estas restrições podem envolver requisitos de desempenho, requisitos de manutenção, requisitos de segurança, requisitos de confiabilidade, requisitos de interoperabilidade e outros tipos de requisitos que estejam ligados à solução como um todo e não a uma funcionalidade específica. Os requisitos não funcionais também são chamados de requisitos de qualidade.

Em sua obra, Vazquez e Simões (2016) afirmam que os requisitos não funcionais complementam a especificação do software. Um software deve funcionar e funcionar bem. Os requisitos funcionais declaram o funcionamento necessário, e os não funcionais

o que é "funcionar bem". Ao esclarecer quais são os critérios para ser considerado um bom funcionamento, os requisitos não funcionais estabelecem os níveis de serviço esperados para aquele software.

Os requisitos não funcionais têm tendência de ter uma importância maior em relação aos requisitos funcionais (KERR,2015). Um usuário comum, na maioria dos casos, tem condições de contornar uma falha em um requisito funcional, mas não tem meios de operar o sistema caso um requisito não funcional que suporta o software esteja indisponível.

Por exemplo: se um relatório de vendas mensal (que é um requisito funcional) não for impresso devido à ocorrência de uma falha no sistema, o usuário provavelmente poderá obter as informações através de uma outra funcionalidade, realizar um "*print*" da tela e apresentar as informações ao seu gestor. No caso de um requisito não funcional, por exemplo, que determina que o software deve funcionar vinculado a um SGDB – Sistema de Gerenciamento de Banco de Dados, e este SGDB estiver indisponível o usuário não terá a possibilidade de realizar nenhuma ação no sistema.

Há uma certa dificuldade inicial entre os estudantes de engenharia de requisitos para diferenciar o que são requisitos funcionais dos não funcionais. Vazquez e Simões (2016) afirmam que "embora não seja uma regra, uma característica que costuma diferenciar os requisitos funcionais dos não funcionais é que os não funcionais costumam se manifestar de forma geral sobre o software, e os funcionais de forma específica".

Beatty e Wiegers (2013, p. 543), chamam a atenção para a relevância dos requisitos não funcionais:

> Devido à ênfase natural na funcionalidade do produto, é fácil negligenciar requisitos não funcionais. Consultar os clientes sobre as características de qualidade, como desempenho, usabilidade, segurança e confiabilidade. Documente esses requisitos não funcionais e seus critérios de aceitação com a maior precisão possível.

Para Sommerville (2018), os clientes e usuários diretos de um sistema têm dificuldades em determinar requisitos não funcionais, pois há necessidade de se ter um direcionamento muito técnico na definição destas questões. Alguns requisitos não funcionais podem ser complexos e difíceis de entender, especialmente para aqueles sem conhecimento técnico. Desta forma, sempre que possível, o analista de requisitos deve envolver pessoal técnico da área de tecnologia daquela organização demandante do software que conhece a realidade da organização e pode ajudar a identificar ou até mesmo determinar os requisitos não funcionais.

Os requisitos não funcionais podem estar ligados a necessidades dos usuários, a restrições orçamentárias, políticas organizacionais, necessidade de interoperabilidade com outros sistemas que a organização já possui, sendo software ou hardwa-

re, ou até mesmo de fatores externos à organização, como normas de segurança de informação como a Lei Geral de Proteção de Dados ou outros regulamentos cabíveis (SOMMERVILLE, 2018).

Sugere-se ainda a utilização da Tabela 1 para que seja possível descrevê-los quantitativamente, pois somente desta forma poderão ser efetivamente validados.

Tabela 1 – Métricas para especificar requisitos não funcionais

| MÉTRICAS PARA ESPECIFICAR REQUISITOS NÃO FUNCIONAIS | |
|---|---|
| **PROPRIEDADE** | **MEDIDA** |
| **Velocidade** | • Transações processadas/segundo<br>• Tempo de Resposta usuário/evento<br>• Tempo de atualização da tela |
| **Tamanho** | • Megabytes<br>• Número de chips de memória ROM |
| **Facilidade de Uso** | • Tempo de treinamento<br>• Número de quadros de Ajuda |
| **Confiabilidade** | • Tempo média para falha<br>• Probabilidade de indisponibilidade<br>• Taxa de ocorrência de falhas<br>• Disponibilidade |
| **Robustez** | • Tempo de reinício após falha<br>• Percentual de eventos que causam falha<br>• Probabilidade de corrupção de dados em caso de falha |
| **Portabilidade** | • Percentual de declarações dependentes no sistema-alvo<br>• Número de sistemas-alvo |

Fonte: Sommerville (2018, p. 94)

Retomando a analogia com a construção civil e o projeto da residência, poderíamos citar como exemplo de requisitos não funcionais para a construção de uma residência:

- As paredes externas devem ser resistentes a chuva e ao vento, sem permitir que a umidade infiltre nas paredes.
- Cada cômodo da residência deve suportar até 15.000kg de carga líquida;
- Deve ser utilizado equipamentos (torneiras, fechaduras, louças) que possuam peças de reposição no mercado da construção civil;
- A casa deve ter boa ventilação e iluminação natural, garantindo a saúde e o conforto dos moradores;
- Os materiais de construção utilizados devem ser duráveis e resistentes à corrosão e ação do tempo;
- A casa deve ser projetada para garantir a privacidade dos moradores, impedindo a visibilidade externa para o interior dos cômodos.

Agora que percebemos que os requisitos não funcionais estão ligados a questões gerais, de confiabilidade e de características gerais da residência, veremos exemplos requisitos não funcionais de um sistema de informação. São exemplos de requisitos não funcionais de um sistema de controle de Frotas:

- Ter disponibilidade de uso 24 horas por dia, 7 dias da semana.
- O tempo de qualquer transação realizada pelo usuário não pode exceder 20 segundos.
- Ser construído utilizando arquitetura de software em 3 camadas, apresentação, negócios e dados.
- Deve possuir uma página de ajuda para cada funcionalidade, com acesso sensível ao contexto.
- O sistema estará disponível pelo menos 99,75% do tempo em dias úteis entre 07:00 e 18:00. Já aos finais de semana, pelo menos 99,1% entre 10:00 e 19:00.
- O sistema deverá ser acessado via browser, inclusive em interface mobile.
- O sistema deve ser capaz de suportar pelo menos 1000 usuários simultâneos sem prejuízo em seu desempenho ou a sua velocidade.
- Os dados dos usuários devem ser armazenados de forma segura e criptografada, seguindo as leis de privacidade e proteção de dados.

Os requisitos não funcionais são tão importantes quanto os requisitos funcionais em um projeto de software. Eles definem as características do sistema, como sua performance, segurança, usabilidade, escalabilidade e outros aspectos que afetam diretamente a experiência do usuário final. É importante que eles sejam identificados e documentados desde o início do projeto, para que sejam incorporados na arquitetura e design do sistema, garantindo sua qualidade e satisfação do usuário.

Uma má definição dos requisitos não funcionais pode levar a uma série de problemas e consequências negativas para o projeto de software. Alguns dos problemas podem incluir (WIEGERS; BEATTY, 2013; SOMMERVILLE, 2018):

- Problemas de desempenho: se os requisitos não funcionais relacionados à performance não forem bem definidos, o sistema pode não atender às expectativas do usuário em termos de velocidade, capacidade de resposta e escalabilidade.
- Problemas de segurança: a falta de consideração dos requisitos não funcionais relacionados à segurança pode levar a vulnerabilidades de segurança que podem ser exploradas por atacantes.
- Problemas de usabilidade: se os requisitos não funcionais relacionados à usabilidade não forem bem definidos, o sistema pode ser difícil de usar e compreender, o que pode afetar a satisfação do usuário e a sua adoção.
- Problemas de escalabilidade: se os requisitos não funcionais relacionados à escalabilidade não forem considerados, o sistema pode não ser capaz de suportar o crescimento do número de usuários ou o aumento da demanda por recursos.

- Problemas de manutenibilidade: a falta de consideração dos requisitos não funcionais relacionados à manutenibilidade pode tornar o sistema difícil de manter e atualizar, o que pode resultar em custos adicionais a longo prazo.

Além disso, a consideração dos requisitos não funcionais ajuda a evitar problemas no futuro, como o aumento de custos para corrigir falhas ou a perda de clientes devido a problemas de desempenho. Portanto, é fundamental que eles sejam considerados na fase de planejamento e sejam monitorados ao longo do desenvolvimento para garantir que o sistema final atenda a todas as necessidades tanto funcionais quanto não funcionais. Enfim, a consideração dos requisitos não funcionais é uma parte crucial para o sucesso de um projeto de software.

## 2.3 REQUISITOS INVERSOS OU RESTRIÇÕES

Requisitos inversos ou restrições são condições declaradas que restringem aspectos do sistema. Estas restrições podem estar vinculadas a aspectos funcionais e não funcionais do software. Uma regra de negócios restritiva determina que certas ações devem ou não devem ser executadas, ou que apenas determinadas pessoas ou perfis podem executar tais ações (WIEGERS; BEATTY, 2013).

Um exemplo de restrição que pode ser vinculada a um requisito funcional:

Requisito funcional: O sistema deve permitir que os usuários adicionem itens ao carrinho de compras.

Restrição (requisito inverso): O sistema não deve permitir que os usuários adicionem um item que está fora de estoque ao carrinho de compras. Além disso, o sistema deve atualizar o estoque em tempo real para evitar que outros usuários tentem comprar o mesmo item que já foi vendido.

Neste exemplo, o requisito funcional especifica que os usuários devem ser capazes de adicionar itens ao carrinho de compras, enquanto o requisito inverso define a restrição de que o sistema deve impedir que usuários adicionem itens fora de estoque ao carrinho de compras para evitar vendas desnecessárias e insatisfação do cliente. O requisito inverso também especifica que o sistema deve atualizar o estoque em tempo real para evitar a venda de itens que já foram vendidos.

Já quando falamos de requisitos não funcionais, geralmente as restrições estão vinculadas a aspectos de performance ou tecnologias. Exemplos:

- O tamanho do aplicativo não deve exceder 30 MB.
- O software deve utilizar apenas bibliotecas dependentes que tenham licença *open source.*

- Nenhuma transação executada no software pode ultrapassar 20 segundos de processamento.

Wiegers e Beatty (2013) ressaltam que as restrições transmitem implicações diretas no desenvolvimento de software e devem ser consideradas pois apresentam de forma complementar aspectos a funcionalidade do software, sejam funcionais – vinculadas ao negócio ou não funcionais – vinculadas a aspectos de ambiente computacional, tecnologia, desempenho, segurança e confiabilidade.

É importante ressaltar que a atividade de descoberta, definição e detalhamento de requisitos é uma atividade intelectual de grande complexidade e que requer uma abordagem sistemática e disciplinada. Para lidar com os desafios destes processos, a revisão de requisitos se apresenta como uma atividade importante no processo de engenharia de requisitos, que envolve a análise sistemática dos requisitos para identificar e corrigir possíveis problemas. A revisão de requisitos ajuda a garantir que os requisitos sejam compreendidos corretamente, que sejam consistentes e completos, e que atendam às necessidades do cliente.

Segundo Pressman e Maxim (2016), a revisão de requisitos é uma atividade crítica no processo de engenharia de requisitos e deve ser realizada por uma equipe multidisciplinar, que inclui membros da equipe de desenvolvimento, clientes e usuários finais. A revisão de requisitos deve ser conduzida em várias etapas do processo de desenvolvimento, incluindo a elicitação de requisitos, a análise e especificação de requisitos e a validação de requisitos.

Durante a revisão de requisitos, a equipe deve avaliar os requisitos quanto à sua clareza, consistência, completude, verificabilidade e rastreabilidade. Isso envolve analisar cada requisito para garantir que ele seja compreendido corretamente, que não haja conflitos ou redundâncias com outros requisitos, e que possa ser verificado e rastreado ao longo do processo de desenvolvimento.

Neste tema, foi possível compreender que a classificação de requisitos em funcionais e não funcionais é um processo essencial na elaboração de um projeto de software. Como resultado dos processos de elicitação de requisitos, eles precisam ser documentados de forma adequada. Enquanto os requisitos funcionais descrevem as funcionalidades e ações que o sistema deve realizar, os requisitos não funcionais especificam como essas ações devem ser realizadas.

A distinção entre esses dois tipos de requisitos é muito importante, pois embora ambos sejam essenciais para o sucesso do projeto, as falhas nos requisitos não funcionais podem ser mais graves e difíceis de corrigir, pois englobarão o projeto como um todo. Os requisitos funcionais têm uma tendência de serem mais fáceis de medir e testar, enquanto os requisitos não funcionais tendem a ser mais subjetivos e difíceis de definir claramente. Por isso, é fundamental que a equipe de desenvolvimento esteja atenta à definição e validação desses requisitos.

É preciso ressaltar que a classificação de requisitos ajuda a equipe de desenvolvimento a priorizar as funcionalidades do sistema, definir metas de desempenho e segurança, e planejar os testes. A classificação de requisitos também ajuda a garantir que os objetivos do projeto estejam alinhados com as necessidades do cliente ou usuário final, resultando em um produto final mais satisfatório e alinhado com as expectativas das partes interessadas.

# RESUMO DO TÓPICO 1

**Neste tema de aprendizagem, você estudou:**

- Os conceitos de requisitos funcionais e não funcionais.
- A importância da documentação dos requisitos funcionais e não funcionais em um projeto de desenvolvimento de software.
- Os tipos de características que podem ser descritas através de requisitos não funcionais.
- Boas práticas para escrita de requisitos funcionais e não funcionais.

# AUTOATIVIDADE

1 Para Pressman e Maxim (2016), os requisitos de software são divididos em duas categorias principais, os funcionais e não funcionais. O que são requisitos funcionais em um projeto de software?

a) ( ) Requisitos que especificam como o software deve se comportar diante de certas situações.
b) ( ) Requisitos que definem as características físicas do software, como tamanho e formato.
c) ( ) Requisitos que estabelecem os critérios de qualidade do software, como velocidade e segurança.
d) ( ) Requisitos que descrevem as interfaces do software com outros sistemas ou dispositivos.

2 Os requisitos não funcionais estão relacionados ao ambiente no qual o software deve permanecer operante. Sobre a alternativa que melhor define o conceito de requisitos não funcionais em um projeto de software, assinale a alternativa CORRETA:

a) ( ) Requisitos que especificam as características físicas do software, como tamanho, formato e disposição dos elementos na tela.
b) ( ) Requisitos que estabelecem os critérios de qualidade do software, como velocidade de processamento, confiabilidade e segurança da informação.
c) ( ) Requisitos que descrevem as interfaces do software com outros sistemas ou dispositivos, incluindo as especificações de integração.
d) ( ) Requisitos que detalham os objetivos comerciais do software, como lucro, crescimento e retorno sobre o investimento.

3 A classificação de requisitos é essencial para organizar um projeto de software e permitir que ele seja construído seguindo as melhores práticas e atendendo os objetivos dos usuários. Sobre os requisitos funcionais, analise as sentenças a seguir:

I- O software deve ser capaz de realizar cálculo de folha de pagamento.
II- O software deve ser capaz de lidar com uma carga de 100 usuários simultâneos.
III- O software deve ser compatível com o sistema operacional Windows.
IV- O software deve ser capaz de emitir um relatório com os valores recolhidos em impostos em cada mês.
V- O software deve ser capaz de armazenar dados de forma segura.

Assinale a alternativa CORRETA:

a) ( ) As sentenças I e III estão corretas..
b) ( ) As sentenças II e III estão corretas.
c) ( ) As sentenças I e IV estão corretas.
d) ( ) As sentenças II e IV estão corretas.

4 Os requisitos funcionais têm ligação com o que o software deve fazer. Os não funcionais estão vinculados a características de qualidade e arquiteturais do software. Assim, crie uma lista de requisitos funcionais no contexto de um sistema de controle de um restaurante delivery.

5 Os requisitos não funcionais atuam para restringir as soluções. Estas restrições podem envolver requisitos de desempenho, requisitos de manutenção, requisitos de segurança, requisitos de confiabilidade, requisitos de interoperabilidade e outros tipos de requisitos que estejam ligados à solução como um todo e não a uma funcionalidade específica. Assim, monte uma lista de requisitos não funcionais de um aplicativo mobile.

# DOCUMENTAÇÃO DE ESPECIFICAÇÃO DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 2, abordaremos a documentação de especificação de requisitos que é um artefato de software que descreve detalhadamente os requisitos funcionais e não funcionais e outros aspectos relevantes em relação a um projeto de software. Este documento é uma parte crítica do processo de desenvolvimento de software, pois fornece uma base clara e concisa para a equipe de desenvolvimento compreender as expectativas e os objetivos do projeto. Além disso, a documentação de especificação de requisitos também é importante para garantir que todas as partes envolvidas, incluindo os clientes, os gerentes de projeto e a equipe de desenvolvimento, estejam alinhadas quanto às expectativas e objetivos do projeto.

A documentação de especificação de requisitos é um documento vivo que evolui ao longo do tempo, à medida que o projeto é desenvolvido e novas informações são conhecidas. Ela é uma referência vital para o projeto, fornecendo informações claras e precisas sobre as funcionalidades esperadas, as restrições de desempenho, as características de usabilidade e outros requisitos importantes. Além disso, a documentação de especificação de requisitos também é importante para garantir a qualidade e a consistência do projeto, pois fornece a base para testes rigorosos e verificações de conformidade.

Além do formato tradicional de documentação de software, também será apresentado as histórias de usuário que são uma forma fácil de entender as necessidades do software, ajudando a equipe de desenvolvimento a compreender o contexto e as motivações dos usuários. Além disso, as histórias de usuário também ajudam a priorizar as funcionalidades e a garantir que o produto final atenda às necessidades reais dos usuários. Elas são uma parte importante do processo de desenvolvimento de software, pois fornecem uma base para o planejamento, a execução e o rastreamento do progresso do projeto.

## 2 DOCUMENTAÇÃO DE ESPECIFICAÇÃO DE REQUISITOS

A documentação de software é uma parte essencial do processo de desenvolvimento de software, pois fornece informações claras e concisas sobre o funcionamento do software, sua estrutura e sua utilização. Alguns dos principais benefícios da documentação de software incluem (SOMMERVILLE, 2018; PRESSMAN; MAXIM, 2016):

- Compreensão do software: a documentação de software ajuda a equipe de desenvolvimento, bem como outras partes interessadas, a compreender como o software funciona e suas funcionalidades.
- Facilidade de manutenção: a documentação de software fornece informações valiosas sobre a estrutura do software, o que torna mais fácil para a equipe de manutenção realizar atualizações e correções.
- Conformidade regulatória: em alguns casos, a documentação de software é necessária para atender aos regulamentos e requisitos legais.
- Transferência de conhecimento: a documentação de software fornece informações valiosas para a equipe de desenvolvimento e outras partes interessadas, o que facilita a transferência de conhecimento de um grupo de pessoas para outro.

A documentação de software pode ser vista como um contrato entre o desenvolvedor e os usuários. Ela descreve as expectativas e requisitos para o software, incluindo suas funcionalidades, limitações, capacidade de suporte e requisitos de sistema. Ao utilizar o software, os usuários concordam em trabalhar dentro das restrições e expectativas estabelecidas na documentação.

Armazenar artefatos de software é um aspecto importante do desenvolvimento de software, pois permite a gestão e preservação dos documentos e informações relevantes para o projeto. Os artefatos de software podem incluir requisitos, diagramas, modelos, especificações, código-fonte, entre outros. Em sua obra, Wiegers e Beatty (2013) afirmam que manter um registro claro e organizado de todos os artefatos de requisitos é essencial para o sucesso do projeto. Se você não pode encontrar um artefato, ou se acredita que ele foi perdido ou corrompido, então não pode confiar no seu conhecimento do que é necessário para o produto.

Uma boa prática para armazenar artefatos de software é utilizar ferramentas de controle de versão, como o Git em gerência de configuração de software, que permitem o controle de alterações e o compartilhamento dos arquivos entre os membros da equipe de desenvolvimento. Além disso, é importante manter uma estrutura organizada e documentada para facilitar o acesso e localização dos artefatos.

## 2.1 TÉCNICAS DE MODELAGEM E ESPECIFICAÇÃO DE REQUISITOS

Para Sommerville (2018), a especificação de requisitos consiste no processo de escrever os requisitos de usuário e de sistema em um documento, denominado documento de requisitos. Na busca pela excelência, os requisitos precisam ser claros, inequívocos, fáceis de entender, completos e consistentes. Alcançar isso na prática é praticamente impossível, pois as partes interessadas interpretam os requisitos de formas diferentes e também há conflitos e incoerências entre eles.

Uma especificação de requisitos deve ser escrita de modo a garantir o conhecimento a todas as partes interessadas daquele software, evitando-se a utilização de jargões de software, e sem incluir detalhes da arquitetura do software. A especificação deve ser independente de tecnologia, deve ser concentrada nos aspectos de negócio (SOMMERVILLE, 2018).

Um aspecto bem importante destacado por Sommerville (2018) é que a especificação dos requisitos do software pode ser utilizada como parte do contrato para desenvolvimento do sistema, e por isso deve contemplar uma especificação completa e detalhada do sistema como um todo.

De acordo com Pressman e Maxim (2016), a especificação de requisitos utilizando linguagem natural é uma abordagem que utiliza uma linguagem clara e compreensível para descrever os requisitos do software. Isso é feito com o objetivo de tornar os requisitos mais acessíveis e compreensíveis para todas as partes interessadas, incluindo os desenvolvedores, gerentes de projeto e clientes. A especificação de requisitos em linguagem natural é uma parte importante do processo de desenvolvimento de software, pois ajuda a garantir que todos estejam alinhados quanto aos objetivos e às expectativas do projeto.

Para minimizar os conflitos de entendimentos ao escrever requisitos utilizando a linguagem natural, Sommerville (2018) faz uma série de recomendações sobre tema, que valem ser consideradas nos projetos de software:

- Construir um formato padrão e garantir que todas as definições de requisitos o utilizem. A padronização proporciona a redução da probabilidade de omissões e permite que os requisitos sejam conferidos de uma forma mais fácil. Sempre que o contexto permitir, os requisitos devem ser escritos em uma ou duas frases no máximo.
- Distinguir os requisitos obrigatórios dos desejáveis, utilizando-se dos termos “deve” e “pode”, respectivamente.
- Utilizar realces como negrito, itálico e cores diferenciados para destacar ou separar partes importantes do requisito.

Estas recomendações devem ser seguidas à risca para que sejam evitadas as falhas mais triviais ao escrever requisitos em linguagem natural (PRESSMAN; MAXIM, 2016; VALENTE, 2020):

- Ausência de clareza: Requisitos vagos ou ambíguos podem levar a interpretações erradas e soluções equivocadas.
- Ausência de especificidade: Requisitos genéricos não fornecem informações suficientes para a implementação correta do software.
- Ausência de testabilidade: Requisitos que não podem ser facilmente testados ou verificados são difíceis de garantir e validar.

- Ausência de priorização: Requisitos não classificados em ordem de importância tornam difícil para o desenvolvedor entender o que é mais importante para o usuário.
- Ausência de consistência: Requisitos contraditórios ou inconsistentes podem levar a soluções confusas e problemas de implementação.

Como mencionado, a linguagem natural oferece uma série de armadilhas para escrita de requisitos. A seguir, serão apresentados uma série de situações comuns que podem ocorrer na escrita de requisitos que são inadequadas e o motivo pelo qual cada uma delas é considerada indevida (SOMMERVILLE, 2018; PRESSMAN; MAXIM, 2016; IEEE, 1998):

- O sistema deve ser fácil de usar: esse requisito não é claro porque não define o que significa "fácil de usar". O que é fácil para um usuário pode não ser fácil para outro.
- O software deve ser rápido: esse requisito não é coeso porque não especifica o que o software deve ser rápido. Rápido em realizar qual tarefa? A velocidade é medida em segundos, milissegundos ou microssegundos?
- O sistema deve ser bonito: esse requisito não é testável porque "bonito" é subjetivo e não pode ser medido objetivamente.
- O software deve ser fácil de instalar: esse requisito não é rastreável porque não especifica como o software deve ser fácil de instalar. Quais etapas de instalação devem ser simplificadas?
- O sistema deve ser amigável: esse requisito não é claro porque não define o que significa "amigável". Amigável com quem? Usuários finais ou desenvolvedores?
- O software deve ser barato: este requisito não é coeso porque não especifica qual aspecto do software deve ser barato. O preço de compra, o custo de manutenção ou o custo de implantação?
- O sistema deve ser intuitivo: esse requisito não é testável porque "intuitivo" é subjetivo e não pode ser medido objetivamente.
- O software deve ser compatível com todos os sistemas operacionais: esse requisito não é rastreável porque não especifica quais sistemas operacionais devem ser suportados e em quais versões.
- O sistema deve ser seguro: esse requisito não é claro porque não especifica quais medidas de segurança devem ser implementadas. O que é considerado seguro em um sistema pode não ser considerado seguro em outro.
- O software deve ser fácil de manter: esse requisito não é coeso porque não especifica quais aspectos do software devem ser fáceis de manter. A manutenção pode incluir atualizações, correções de bugs, entre outros.

A especificação detalhada de requisitos é uma etapa crucial na concepção de um sistema, pois permite que as expectativas e necessidades do usuário sejam claramente definidas e entendidas pelo time de desenvolvimento. Esta especificação deve incluir tanto os requisitos funcionais quanto os requisitos não funcionais do sistema (IEEE, 1998).

Uma especificação detalhada de requisitos deve incluir informações como (IEEE, 1998):

- Objetivos do sistema: o que o sistema deve abranger e como deve atender às necessidades do usuário.
- Requisitos funcionais: as funcionalidades específicas que o sistema deve fornecer.
- Requisitos não funcionais: critérios de controle e estrutura, como desempenho, segurança, usabilidade, disponibilidade, conformidade, integração e manutenibilidade.
- Restrições: quaisquer limitações ou restrições que possam afetar o projeto.
- Prioridades: quais são as funcionalidades e requisitos mais importantes e qual o grau de prioridade de cada um.
- Prazos: o cronograma previsto para o desenvolvimento do sistema e a entrega das funcionalidades.
- Usuários: quem serão os usuários finais do sistema e como eles interagirão com ele.
- Ambiente: o ambiente em que o sistema será executado, incluindo hardware, software e requisitos de rede.

Para que seja melhor compreendido a estrutura de um documento de especificação de requisitos, a seguir será apresentado um exemplo de especificação de requisitos de um sistema escolar:

**Especificação de requisitos do Sistema de Gestão Escolar**

**Objetivos do sistema:**

O objetivo deste sistema é fornecer uma plataforma de gerenciamento completo para uma escola, automatizando todos os processos administrativos e acadêmicos. O sistema deve permitir a gestão de informações dos alunos, funcionários e professores, além de fornecer um ambiente de ensino virtual e facilitar a comunicação entre a escola e os pais.

**Requisitos funcionais:**

- RF01 - O sistema deve permitir o cadastro de alunos, funcionários e professores.
- RF02 - O sistema deve permitir o cadastro de turmas, disciplinas e notas.
- RF03 - O sistema deve permitir a emissão de boletins escolares.
- RF04 - O sistema deve permitir a gestão de informações financeiras, como mensalidades e pagamentos.
- RF05 - O sistema deve permitir a criação de um ambiente de ensino virtual, com aulas on-line e atividades para os alunos.
- RF06 - O sistema deve permitir a comunicação entre a escola e os pais, através de um canal de mensagens.
- RF07 - O sistema deve permitir a geração de relatórios de desempenho acadêmico dos alunos.
- RF08 - O sistema deve permitir a visualização de horários de aulas e de disponibilidade de professores.

- RF09 - O sistema deve permitir o controle de acesso e permissões para usuários do sistema.
- RF10 - O sistema deve permitir o armazenamento e gerenciamento de documentos importantes da escola, como atas de reuniões e documentos administrativos.

**Requisitos não funcionais:**

- RNF01 - O sistema deve possuir um design de interface amigável e intuitivo, permitindo que usuários de diferentes níveis de habilidade o utilizem facilmente.
- RNF02 - O sistema deve adotar medidas de segurança, como criptografia e autenticação de usuários, para proteger as informações pessoais dos usuários, como nome, CPF, endereço e histórico escolar.
- RNF03 - O sistema deve ser capaz de se adaptar a diferentes tamanhos de tela e dispositivos, como desktops, tablets e smartphones, oferecendo uma experiência consistente em todas as plataformas.
- RNF04 - O sistema deve ser construído com uma arquitetura flexível e modular, permitindo a adição de novas funcionalidades ou integrações com outros sistemas no futuro, sem comprometer a estabilidade e o desempenho do sistema.
- RNF05 - O sistema deve ser capaz de lidar com grande volume de acessos simultâneos, garantindo a escalabilidade e a disponibilidade do sistema em momentos de pico, como no início das aulas ou em épocas de matrícula.

**Restrições**:

- O sistema deve ser desenvolvido em linguagem de programação Java, utilizando o framework Spring Boot.
- O sistema deve ser hospedado em um servidor Linux, utilizando banco de dados PostgreSQL.
- O prazo para o desenvolvimento do sistema é de 6 meses, com uma versão mínima viável a ser entregue em 3 meses.
- O sistema deve ser utilizado por alunos, professores, coordenadores e funcionários da escola, totalizando um número estimado de 500 usuários ativos.

**Prioridades:**

- Os requisitos relacionados à gestão de informações dos alunos são considerados prioritários, pois são cruciais para o bom funcionamento da escola.
- Os requisitos relacionados ao ambiente de ensino virtual são considerados de alta prioridade, pois possibilitam o ensino a distância.
- Os requisitos relacionados à comunicação entre a escola e os pais são considerados de média prioridade, pois podem ser realizados por outros meios.

**Prazos:**

O sistema deve estar totalmente implementado e testado até o final do semestre letivo atual.

Os testes do sistema devem ser realizados por um período de 30 dias antes da implantação.

**Usuários:**

Os usuários do sistema são os administradores da escola, professores, alunos e pais/responsáveis.

**Ambiente:**

O ambiente em que o sistema será utilizado é a escola e seus usuários terão acesso através de um navegador web.

É importante ressaltar que um modelo de especificação de requisitos deve ser adaptável a diferentes tipos de sistemas, projetos e necessidades dos usuários (SOMMERVILLE, 2018). O modelo deve ser capaz de se adaptar às mudanças nas necessidades do projeto, bem como permitir a inclusão de novos requisitos conforme necessário. Essa abordagem flexível permite que o modelo de especificação de requisitos seja aplicado a uma variedade de projetos de software, independentemente de seu tamanho ou complexidade. Além disso, uma abordagem flexível também ajuda a garantir que os requisitos sejam suficientemente detalhados e precisos para orientar o desenvolvimento do sistema.

O modelo de especificação de requisitos é flexível porque pode ser adaptado às necessidades de cada projeto. Isso significa que o modelo pode ser ajustado para se adequar à complexidade do sistema, ao tamanho da equipe de desenvolvimento e a outros fatores relevantes. Além disso, o modelo pode ser utilizado em diferentes tipos de projetos de software, como projetos de grande porte ou projetos mais simples.

A flexibilidade do modelo de especificação de requisitos também permite que ele seja atualizado à medida que o projeto evolui. À medida que novas informações surgem ou novos requisitos são adicionados, o modelo pode ser modificado para incluir essas mudanças. Isso ajuda a garantir que o sistema final seja entregue com todos os requisitos necessários.

Outra vantagem da flexibilidade do modelo de especificação de requisitos é que ele permite que a equipe de desenvolvimento se adapte às mudanças no ambiente externo. Por exemplo, se houver uma mudança nas expectativas do cliente ou nas condições do mercado, a equipe de desenvolvimento pode ajustar o modelo para refletir essas mudanças e garantir que o sistema final atenda às novas necessidades.

A documentação de especificação de requisitos é uma parte fundamental do desenvolvimento de software. Ela permite que os desenvolvedores compreendam claramente os requisitos do usuário, garantindo que o software seja construído corretamente e atenda às expectativas dos usuários. A documentação também ajuda a evitar problemas futuros, como mudanças imprevistas no software ou conflitos entre as equipes de desenvolvimento. Além disso, ela fornece uma referência para futuras equipes de manutenção e suporte. Em suma, a documentação de especificação de requisitos é uma peça-chave para garantir o sucesso de um projeto de software e é altamente recomendável investir tempo e esforço para criar uma documentação completa e precisa (SOMMERVILLE, 2018).

## 2.2 HISTÓRIAS DE USUÁRIO

Histórias de usuário são uma forma de documentar funcionalidades de sistemas desenvolvidos por equipes que aplicam os conceitos do Manifesto Ágil. Portanto, suas características se enquadram nos valores e princípios do Manifesto.

O Manifesto Ágil é composto por 12 princípios e pela declaração de quatro valores, incluindo o valor a seguir, que é um dos mais importantes e discutidos pelos engenheiros de software: software funcionando tem mais valor que uma documentação abrangente (BECK *et al*., 2001).

Muitas pessoas interpretam esta declaração como uma autorização que o manifesto ágil dá para ausência de documentação, mas não é isso que o Manifesto Ágil preconiza. O Manifesto propõe a valorização de um software entregue funcionando com uma maior prioridade em relação a uma documentação extensa, não relevante, burocrática e ineficiente e por muitas vezes um software que não está pronto.

Visando este contraponto, as histórias de usuário são a alternativa proposta para a documentação das necessidades dos usuários de forma ágil. A proposta é envolver os usuários na elaboração e validação dessa documentação, colaborando, também, com um dos princípios ágeis: os fornecedores de requisitos e desenvolvedores devem trabalhar em conjunto frequentemente, durante todo o projeto (BECK *et al*., 2001).

A seguir, será apresentado as histórias de usuário: sua estrutura, exemplos e os critérios para elaboração de uma história de usuário adequada.

Kent Beck foi o responsável por introduzir o termo histórias de usuário pela primeira vez, como parte da *Metodologia Extreme Programming* (XP). O objetivo foi promover um estilo mais informal e de conversação para a elicitação de requisitos em contraponto a longas especificações escritas em um formato muito formal e detalhado. Beck percebeu que muitas vezes os requisitos detalhados em documentos extensos e

complexos não eram compreendidos pelos usuários finais ou pelos desenvolvedores, e que esses documentos muitas vezes não eram atualizados conforme as mudanças ocorriam no projeto (FOWLER, 2013).

De acordo com Cohn (2004), uma história de usuário é uma técnica utilizada em metodologias ágeis de desenvolvimento de software, que consiste em descrever uma funcionalidade do software de uma maneira simples e compreensível para o usuário final, representando uma necessidade real do negócio. Ela é escrita em linguagem natural e é composta por uma breve descrição da funcionalidade, seus critérios de aceitação e a identificação do usuário ou grupo de usuários que se beneficiará dela. Essa abordagem permite que o cliente participe ativamente do processo de desenvolvimento, fornecendo informações valiosas sobre suas necessidades e expectativas em relação ao produto final, e ajuda a equipe a manter o foco em criar valor para o negócio, em vez de se prender a requisitos técnicos complexos e pouco relevantes. Dessa forma, as histórias de usuário permitem uma maior flexibilidade na definição dos requisitos, tornando o processo de desenvolvimento mais adaptativo e colaborativo.

Segundo Rehkopf (2023), as histórias de usuários apresentam vários benefícios importantes, como o enfoque no usuário. Enquanto uma lista de tarefas direciona a equipe para as atividades a serem concluídas, uma coleção de histórias direciona a equipe para a resolução de problemas dos usuários reais.

A colaboração também é oferecida pelas histórias, pois com um objetivo final claro, a equipe pode trabalhar em conjunto para determinar a melhor maneira de atender o usuário e alcançar esse objetivo. Além disso, as histórias estimulam soluções criativas. Elas incentivam a equipe a pensar criticamente e criativamente sobre a melhor maneira de alcançar o objetivo final. Outro fator importante, é que as histórias criam uma cadência. À medida que a equipe passa por cada história, experimenta um pequeno desafio e vitória, impulsionando o ímpeto de desenvolvimento, isso é uma característica do ser humano, ser movido por conquistas (REHKOPF, 2023).

Os três aspectos que compõem uma história de usuário são: cartão, conversação e confirmação, apresentados em detalhes a seguir (JEFFRIES, 2001).

O cartão é parte que contém a descrição escrita da história. É utilizada como um lembrete e para ações de planejamento. Segundo Cohn (2004), o cartão de história é uma descrição escrita da funcionalidade que o cliente deseja. É uma promessa de uma conversa posterior. Ou seja, o cartão da história de usuário é uma forma de iniciar o diálogo entre o time de desenvolvimento e o cliente, de modo a definir e esclarecer os requisitos funcionais do software. Além disso, Cohn (2004) propôs um padrão de escrita para os cartões de histórias de usuário, em que ele sugere que o texto seja divido em três partes, conforme apresentado na Figura 1.

Figura 1 – Padrão do Cartão de História de Usuário


Fonte: adaptado de Cohn (2004)

A primeira parte da história (COMO UM <TIPO DE USUÁRIO>) tem como objetivo deixar bem especificado quem é o beneficiário daquela história. O beneficiário da história é quem vai obter os proveitos de tê-la implementada. É muito importante personificá-lo, no lugar de utilizar, simplesmente, "o usuário", pois todo mundo que usa um sistema é um usuário, então, por meio da personificação de que tipo é aquele usuário, é oportunizado ter uma história melhor escrita e melhor entendida.

Desta forma, é possível entender melhor a razão para qual aquele tipo de usuário precisa daquela história, definir melhor o nível de granularidade das informações a serem processadas na história. Por exemplo, um usuário Gerencial tem um foco diferente de um Operacional dentro de um sistema. O usuário Gerencial busca informações mais compiladas e voltadas a tomada de decisão. Já o usuário operacional, precisa obter/informar detalhes das transações do dia a dia de uma companhia. Não devemos ficar apenas atrás de uma definição de um título de trabalho. A equipe deve ter uma compreensão compartilhada de quem é aquele tipo de usuário.

Veja como a história fica mais completa/rica se você utilizar o papel do usuário de forma contextualizada:

- "Como um cliente..."
- "Como um contribuinte..."
- "Como um funcionário..."
- "Como um gerente de vendas..."
- "Como um professor..."

A segunda parte do cartão é a história em si, o que realmente deve ser desenvolvido. Nessa parte da história, é muito importante deixar claro o que o usuário necessita, o que ele precisa que o software disponibilize para que ele execute as tarefas, o trabalho dele. Essa parte é essencial para garantir que o time de desenvolvimento entenda exatamente o que o usuário espera que o sistema faça. É importante que a descrição seja clara e concisa, para evitar mal-entendidos ou interpretações errôneas. Por exemplo:

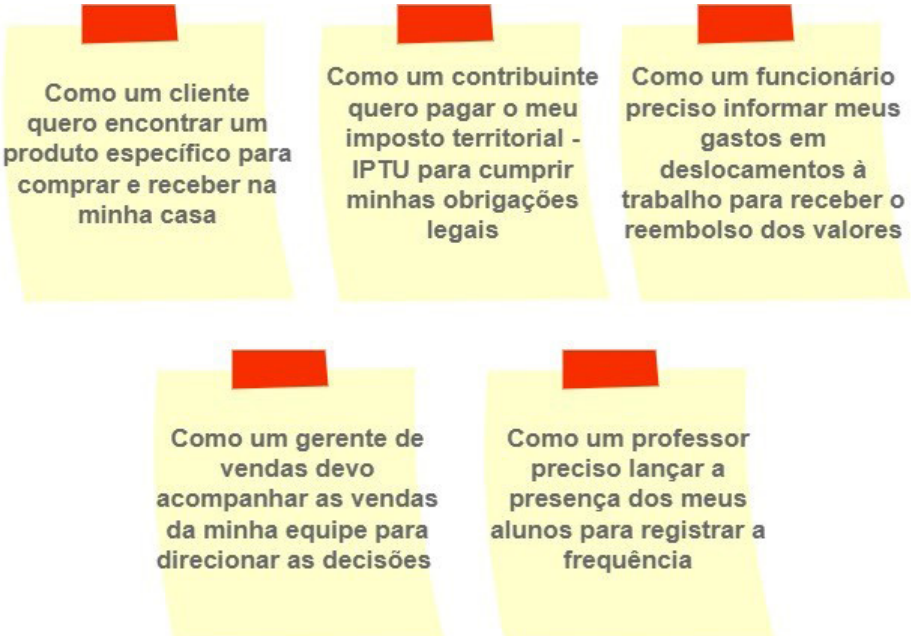
- “Quero encontrar um produto específico”.
- “Quero pagar o meu imposto territorial – IPTU”.
- “Preciso informar meus gastos em deslocamentos à trabalho”.
- “Devo acompanhar as vendas da minha equipe”.
- “Preciso lançar a presença dos meus alunos”.

A terceira parte do cartão, por sua vez, mostra o valor da história, seu objetivo, porque essa história será feita e qual é sua importância, seu valor para o negócio, para o problema que ela está resolvendo. Por exemplo:

- “Para comprar e receber em casa”.
- “Para cumprir minhas obrigações legais”.
- “Para receber o reembolso dos valores”.
- “Para direcionar as decisões”.
- “Para registrar a frequência”.

Agora, veja, na Figura 2, os cartões destas histórias com sua estrutura completa (beneficiário, necessidade e resultado).

Figura 2 – Exemplos de cartões de histórias de usuário




Fonte: o autor

Dando continuidade no entendimento da estrutura de uma história de usuário, é o momento de compreender a conversação. A colaboração com o cliente é um dos princípios das metodologias ágeis e é uma das grandes dificuldades no processo de levantamento de requisitos, junto ao entendimento das necessidades do cliente.

A conversação pode ajudar a superar esses obstáculos, pois promove a interação, faz com que os usuários se sintam mais à vontade em utilizar sua própria linguagem para discutir as histórias. Esses momentos ocorrem com o tempo, em rodadas para refinamento das histórias, quando a história é estimada e também quando a história é discutida para ser incluída no planejamento de entrega, quando ela será selecionada para o desenvolvimento.

A ideia é que as conversas sejam amplamente verbais, mas, sempre que necessário e ou possível, possam ser registradas em documentos. Modelos, templates e exemplos são sempre bem-vindos. O mais importante é que, a partir desse conjunto de conversas, sejam preparados os critérios de aceite para a história, que são a confirmação do valor daquela história para o usuário.

A Confirmação é a estrutura que permitirá a validação da história, ou seja, se ela atende às necessidades do seu beneficiário.

Para complementar a descrição principal da história, e permitir que a confirmação seja realizada, baseado nos resultados da conversação, devem ser descritos os critérios de aceitação da história de usuário. Trata-se de uma lista de itens de negócio que expressam a forma de utilizar a funcionalidade implementada na história.

Esses critérios vão desde regras de negócio até requisitos de interface. Esses critérios servirão como base para a escrita dos testes de aceitação.

Exemplo de critérios de aceitação:

- "O cliente só poderá verificar seu extrato se estiver autenticado".
- "O sistema deverá demonstrar os gastos do titular e adicionais".
- "Os produtos devem ser apresentados em ordem alfabética".
- "Se a declaração já tiver sido enviada no ano corrente, o tipo deve ser Retificação".
- "Não deve ser permitido informar mais de uma despesa com hospedagem no mesmo dia".

Para escrita dos critérios de aceite de histórias de usuários, é possível utilizar a técnica BDD – Behavior-Driven Development –, que significa Desenvolvimento Guiado por Comportamento. É uma abordagem de desenvolvimento de software que se concentra na entrega de valor aos negócios, colaboração entre equipe e comunicação clara.

Segundo Adzic (2011), BDD é uma abordagem de desenvolvimento de software que se concentra em compreender o comportamento desejado do sistema e traduzi-lo em especificações concisas e compreensíveis para todos os stakeholders.

O BDD envolve a criação de exemplos de uso ou cenários baseados em comportamentos para descrever o comportamento esperado do sistema. Esses exemplos são escritos em linguagem natural, tornando-os fáceis de compreender para pessoas que não têm conhecimento técnico.

Aqui está um exemplo de BDD para um sistema de compras on-line:

**História de Usuário**: como um cliente, eu quero adicionar um produto ao carrinho de compras para que eu possa comprá-lo.

**Critérios de Aceite utilizando BDD**
Cenário: Adicionar um produto ao carrinho de compras
Dado que o usuário esteja navegando na loja on-line
Quando o usuário clicar no botão "Adicionar ao carrinho" para um produto específico
Então, o produto deve ser adicionado ao carrinho de compras do usuário
O número de itens no carrinho deve ser atualizado
A página deve exibir uma mensagem confirmando que o produto foi adicionado ao carrinho

Cenário: Concluir a compra com sucesso
Dado que o usuário tenha adicionado pelo menos um produto ao carrinho de compras
Quando o usuário clicar no botão "Finalizar compra" e preencher todas as informações necessárias, incluindo endereço de entrega e método de pagamento
Então, a compra deve ser concluída com sucesso
O usuário deve receber uma confirmação por email
A página deve exibir uma mensagem de sucesso confirmando a compra.

Estes exemplos de cenários de comportamento fornecem uma base clara para o desenvolvimento e teste da história de usuário relacionada a compras on-line, garantindo que o comportamento esperado seja entregue aos usuários.

A escrita de histórias inclui a usabilidade, pois, como afirma Campos (2011), essa é uma forma de garantir a eliminação de ambiguidades, e permite o fornecimento de material para a equipe de desenvolvimento conhecer a funcionalidade do ponto de vista do usuário e ter a possibilidade de confirmar que a história está completa, funcionando e atendendo as necessidades dos usuários.

Incluir aspectos de usabilidade em histórias de usuário é importante para garantir que o sistema seja fácil e intuitivo de usar. A usabilidade é a medida da satisfação e eficiência dos usuários ao usar um sistema (NIELSEN, 1993). Quando aspectos de usabilidade são incluídos nas histórias de usuário, a equipe de desenvolvimento tem uma melhor compreensão de como o sistema deve ser projetado e implementado para ser fácil de usar.

Exemplo:

**História de usuário**: como um cliente de supermercado, quero ser capaz de passar as compras rapidamente e facilmente na caixa de pagamento.

**Critérios de usabilidade incluídos**:

A tela de pagamento deve ser clara e fácil de ler.
O teclado deve ter botões grandes e fáceis de apertar.
O cliente deve ser capaz de escolher a forma de pagamento desejada facilmente.
O cliente deve ser notificado de forma clara quando o item é escaneado.
O cliente deve ser capaz de cancelar ou corrigir itens no carrinho a qualquer momento durante o processo.
A caixa deve exibir o total corrente da compra e a quantidade restante de dinheiro devido.

Esses aspectos de usabilidade garantem que o processo de pagamento seja fácil e intuitivo para o cliente, maximizando a satisfação do usuário e tornando a experiência de compras mais agradável.

Existem diversas maneiras de potencializar o uso de histórias de usuário, em que podemos destacar:

- Escrever histórias de usuário em formato de diálogo: Isso permite uma melhor compreensão das interações entre o usuário e o sistema. Por exemplo, "Como usuário, eu quero poder fazer uma pergunta ao sistema e obter uma resposta precisa".
- Incluir exemplos práticos de diálogo: Isso ajuda a esclarecer como a história de usuário será implementada na prática. Por exemplo, "Usuário: Qual é a previsão do tempo para amanhã? Sistema: Sol pela manhã e nublado à tarde".
- Usar diálogos para priorizar histórias de usuário: Isso ajuda a identificar qual história de usuário é mais importante e deve ser priorizada. Por exemplo, "Como usuário, eu quero poder fazer uma pergunta ao sistema e obter uma resposta precisa, porque isso me permitirá tomar decisões informadas".
- Usar diálogos para identificar possíveis problemas: Isso ajuda a identificar problemas que podem surgir durante a implementação da história de usuário. Por exemplo, "Usuário: Qual é a temperatura atual? Sistema: Desculpe, não estou conseguindo acessar a informação de temperatura".

As histórias de usuário são um meio de comunicação entre a linguagem do mundo de negócios e a linguagem do mundo de tecnologia. Não podemos esperar que um administrador, um advogado, um médico ou contador que esteja envolvido em um processo de definição de uma solução tecnológica tenha conhecimento da linguagem dos desenvolvedores do software, dos termos técnicos que utilizam, e nem é esperado que a equipe de desenvolvimento tenha um pleno conhecimento dos termos dos negócios das soluções que desenvolve, os famosos jargões.

Para isso, a ligação entre usuários e equipe de desenvolvimento necessita ser bem construído. Com o objetivo de guiar e instruir como classificar uma história de usuário como exemplar, bem construída, Wake (2017) propôs o acrônimo em inglês INVEST, que recomenda que uma história para ser considerada excelente deve ser:

**I**ndependent (independente).
**N**egotiable (negociável).
**V**aluable (valiosa).
**E**stimatable (estimável).
**S**mall (pequena).
**T**estable (testável).

A seguir, você verá detalhadamente cada uma das propriedades esperadas em uma história de usuário (WAKE, 2017):

**I**ndependent (Independente**)**: uma história de usuário deve existir independentemente das outras. Isso significa que uma história não deve depender de outras, nem outras devem depender dela. Assim, o responsável por priorizar as histórias pode determinar que elas sejam desenvolvidas em qualquer ordem. Na prática, essa é uma característica muito difícil de se conseguir, pois é muito comum que as histórias tenham pré-requisitos. A ideia é levar o conceito como um norte e sempre chegar o mais próximo possível disso.

Para exemplificarmos esta característica, observe a seguinte história de usuário:
"Como um usuário, eu quero poder fazer login usando minha conta do Facebook ou do Google, para que eu possa acessar o aplicativo facilmente sem ter que me lembrar de mais um nome de usuário e senha".

Embora essa história de usuário atenda a outros critérios do conceito INVEST, pois é valiosa, estimável e testável, ela não é independente, pois depende da integração do aplicativo com as APIs do Facebook e do Google. Se essas integrações não forem concluídas, a história de usuário não poderá ser implementada.

Para tornar essa história de usuário independente, uma opção seria dividir a história em duas histórias menores: uma para implementar o login do Facebook e outra para implementar o login do Google. Dessa forma, cada história seria independente e poderia ser implementada separadamente, garantindo assim esta característica.

**N**egotiable (Negociável)**:** uma história é negociável quando ela foi construída por meio de um processo de colaboração entre os usuários e os desenvolvedores do software, de forma que seus detalhes foram evoluindo em um processo de descoberta e parceria entre os envolvidos no processo.

Veja o exemplo a seguir para avaliarmos se uma história de usuário é negociável ou não:

"Como um usuário, eu quero que o aplicativo exiba um tutorial animado sobre como usar a função X assim que eu faço login pela primeira vez".

Embora essa história de usuário possa ser valiosa para o usuário final, ela pode não ser negociável porque a equipe de desenvolvimento pode ter restrições técnicas ou de recursos que tornem difícil ou impossível implementar um tutorial animado.

Para tornar essa história de usuário mais negociável, pode ser necessário fazer alterações na funcionalidade específica que o usuário está solicitando. Por exemplo, a equipe de desenvolvimento pode sugerir um tutorial não animado ou um tutorial por escrito que seja mais fácil de implementar. Porque na verdade, o valor que o usuário deseja obter desta história é aprender a usar as funcionalidades. A forma com que isso é feito (animado) é uma sugestão dele. No entanto, o que mais importa é ele aprender a usar a funcionalidade.

**V**aluable (Valiosa)**:** isso significa que a história de usuário precisa ter alto valor para o usuário. A ideia é criar apenas histórias de usuário que entreguem valor para o usuário. O que significa entregar valor? Significa criar soluções que tragam benefícios para os usuários nos processos que eles executam.

Avaliando este aspecto, um exemplo de uma história de usuário que não é valiosa seria:

"Como um vendedor, eu quero que o aplicativo tenha um botão de cor amarela, para que ele fique mais bonito".

Embora essa história de usuário possa ser fácil de implementar e testar, ela não é valiosa porque não fornece nenhum valor real ao usuário final. Não agrega nada aos negócios, ao dia a dia de trabalho daquele usuário. Talvez tenha que se aprofundar o entendimento com ele do porquê ele está sugerindo isto. Que situação o fez pensar nesta necessidade, e então de repente ele pode explicar que pensou no botão desta cor para que chame mais atenção, e deixou claro que é o botão para lançar o endereço do cliente, que por muitas vezes no sistema que ele usa hoje ele esqueceu. Então reescrevendo esta história adicionando este contexto, ela poderia ficar assim:

“Como um vendedor, eu quero que o aplicativo tenha um botão de cor destacada e de fácil acesso para informar os dados de endereço de um cliente, para que eu possa realizar uma venda com mais rapidez e eficiência".

Essa história de usuário agora fornece valor real ao usuário final (o vendedor), pois torna o processo de venda mais rápido e eficiente, ao mesmo tempo que mantém o foco na funcionalidade específica desejada (informar os dados de endereço de um cliente). Além disso, a história é independente, negociável, pequena e testável, pois pode ser facilmente estimada e implementada pela equipe de desenvolvimento.

**E**stimatable (Estimável)**:** para uma história ser estimável, é necessário que ela seja clara e, ao mesmo tempo, detalhada, para que a equipe tenha condições de estimar o trabalho que será necessário para entregar aquela história. Se a equipe não consegue

estimar a história, é porque ela ainda precisa ser mais negociada com o cliente, visando entender todos os detalhes que permitirão que a equipe faça a estimativa da história.

Para avaliarmos o que é uma história estimável, trazemos um exemplo de uma história de usuário que não é estimável:

"Como um vendedor, eu quero que o aplicativo seja o mais rápido possível".

Essa história de usuário não é estimável porque não especifica quais aspectos do aplicativo devem ser aperfeiçoados para torná-lo mais rápido. É muito vaga e não oferece informações suficientes para que a equipe de desenvolvimento possa estimar adequadamente o trabalho necessário para implementar a história.

Para tornar essa história de usuário estimável, é necessário fornecer informações mais detalhadas sobre o que significa "mais rápido" de acordo com as necessidades dos usuários.

Um exemplo de como essa história de usuário pode ser reescrita e tornada estimável é:

"Como um vendedor, eu quero que o tempo de carregamento da página inicial do aplicativo ocorra em menos de três segundos, para que eu possa acessar rapidamente as informações de que preciso".

Agora, essa história de usuário é mais específica e detalhada sobre o que precisa ser feito para tornar o aplicativo mais rápido. Ele fornece um objetivo claro e mensurável que a equipe de desenvolvimento pode usar para estimar o trabalho necessário e definir as etapas específicas para atingir esse objetivo.

**S**mall (Pequena): uma história precisa ser pequena para que ela possa ser entregue em um tempo curto. O grande desafio é criar uma história pequena para contemplar essa característica, mas que contenha todas as outras características para ser uma boa história de usuário. Muitas vezes, ao tentar reduzir o tamanho de uma história quebrando-a em várias, pode-se correr o risco de tornar alguma das partes tão pequena que acaba não tendo valor para o usuário.

Para que possamos entender melhor este contexto, apresentamos um exemplo de uma história de usuário que não atende este critério de tamanho, ou seja, é muito grande.

"Como um funcionário, eu quero que o aplicativo me permita criar e gerenciar eventos, incluindo detalhes de data, hora, localização, convidados e RSVP".

Essa história de usuário é muito grande e envolve muitas funcionalidades diferentes. Ela não pode ser implementada em um curto período de tempo e não é algo que possa ser entregue rapidamente para o usuário final. Além disso, a história de usuário é muito genérica e não fornece informações específicas sobre quais funcionalidades devem ser incluídas ou como a implementação deve ser feita.

Para tornar essa história de usuário pequena e mais gerenciável, é necessário dividi-la em tarefas menores e mais específicas. Por exemplo:

Como um usuário, eu quero poder adicionar um novo evento ao calendário, especificando data, hora e localização.

Como um usuário, eu quero poder adicionar convidados a um evento e enviar convites para eles.

Como um usuário, eu quero poder gerenciar a lista de convidados de um evento e ver quem confirmou presença e quem não pode comparecer.

Ao dividir a história de usuário em tarefas menores e mais específicas, a equipe de desenvolvimento pode implementar cada uma delas separadamente, entregando valor incrementalmente para o usuário final. Além disso, cada tarefa pode ser estimada e planejada com mais precisão, ajudando a evitar atrasos ou dificuldades na entrega.

**T**estable (Testável): Uma história de usuário precisa ser testável, seja por quem desenvolve ou por quem a solicitou. Os critérios de aceitação precisam estar bem descritos para que a história possa ser validada. Se uma história não pode ser testada, é muito provável que ela não tenha atendido a outras características, ou é pequena demais, ou é grande demais ou não entrega valor para os usuários.

Para ficar mais claro, apresentamos um exemplo de história de usuário que não é testável:

"Como um gerente de projetos, eu quero que o aplicativo me ajude a gerenciar meus projetos".

Essa história de usuário é muito ampla e não fornece informações suficientes para permitir que a equipe de desenvolvimento crie testes adequados. Não está claro quais funcionalidades ou recursos específicos devem ser implementados para ajudar o usuário a gerenciar seus projetos, nem como será medido o sucesso da implementação.

Para tornar essa história de usuário testável, é necessário fornecer informações mais detalhadas sobre como o aplicativo pode ajudar o usuário a gerenciar seus projetos e quais funcionalidades específicas são necessárias. Por exemplo:

"Como um gerente de projetos, eu quero que o aplicativo me permita criar tarefas e atribuí-las a projetos específicos, para que eu possa acompanhar o progresso dos meus projetos".

Essa história de usuário agora inclui uma funcionalidade específica que pode ser testada para verificar se foi implementada corretamente. A equipe de desenvolvimento pode criar testes para verificar se o usuário pode criar tarefas e atribuí-las aos projetos corretos, e se essas tarefas são exibidas corretamente no aplicativo.

Quando se trata de histórias de usuário, os critérios INVEST são complementares entre si, o que significa que quando uma história não atende a um critério, ela provavelmente também não atende a um ou a todos os outros critérios. Por exemplo, se uma história não é negociável, ela também pode não ser valiosa ou estimável. Da mesma forma, se uma história não é pequena, ela pode não ser testável ou independente.

Por isso, é importante considerar todos os critérios INVEST ao escrever histórias de usuário, pois eles se complementam e se inter-relacionam. Por exemplo, uma história de usuário que é valiosa, mas não é estimável, pode ser difícil de implementar e pode levar a atrasos na entrega. Da mesma forma, uma história de usuário que é pequena, mas não é testável, pode não fornecer valor real ao usuário final, pois não é possível verificar se a funcionalidade está funcionando corretamente.

Ao usar os critérios INVEST em conjunto, a equipe de desenvolvimento pode garantir que as histórias de usuário sejam gerenciáveis, entregáveis e atendam às necessidades do usuário final. Além disso, ao garantir que todas as histórias de usuário atendam aos critérios INVEST, a equipe pode reduzir o risco de atrasos e erros na implementação, garantindo assim o sucesso do projeto como um todo.

O processo de criação de histórias de usuário pode acontecer de várias formas, a depender do tipo de software que está sendo construído. Por exemplo, se o produto a ser desenvolvido é um software sob demanda, ou seja, será desenvolvido especificamente para aquele cliente, ou se está sendo desenvolvido um produto padrão, que vai ser utilizado por muitos clientes, em diversos cenários, sem ter uma empresa ou organização específica como alvo as abordagens poderão ser diferentes.

No primeiro caso, do software para uma empresa específica, as histórias de usuário, devem ser criadas considerando principalmente os aspectos e necessidades daquela organização que o solicita, já que o propósito dele é atender as necessidades particulares da organização para o qual ele está sendo construído.

Já no segundo caso, que antigamente era chamado de software de prateleira, porque ele era gravado em uma mídia física (Cd) e vendido para qualquer um que quisesse adquiri-lo, hoje, este tipo de solução existe no formato Software as a Service (SaaS), em que a empresa desenvolvedora fornece este software para diversas empresas que tem necessidade de automatizar aquela área de negócios. Neste cenário, é importante ter um olhar mais voltado ao negócio, os regulamentos e rotinas comuns que aquele tipo de negócio possui.

## NOTA

Software as a Service (SaaS) é um modelo de entrega de software no qual a aplicação é fornecida pela internet, geralmente como um serviço pago, em vez de ser instalada localmente no computador de cada usuário. O software é acessado via uma conexão à internet a partir de qualquer lugar, a qualquer momento, sem a necessidade de instalar ou configurar o software em seu dispositivo.

Entre as partes interessadas que podem ser envolvidas nas construções de histórias de usuário, podemos destacar (COHN, 2004):

- Usuários finais: são os principais beneficiários do software, portanto, é importante obter suas contribuições para garantir que as funcionalidades correspondam às suas necessidades.
- Equipe de desenvolvimento: como responsáveis por implementar as histórias de usuário, é importante que eles tenham uma compreensão clara do será esperado deles.
- Donos do produto: Tem a responsabilidade de definir a estratégia do produto e garantir que as histórias de usuário estejam alinhadas a ela.
- Stakeholders de negócios/Consultores de Negócio: Eles podem ter uma visão geral das necessidades do negócio e podem fornecer informações valiosas sobre como o software pode ser usado para atender a essas necessidades.
- Gerentes de projeto: Podem fornecer uma perspectiva sobre como as histórias de usuário se encaixam no contexto do projeto e podem ajudar a priorizar as funcionalidades mais importantes.

O que não se pode, em nenhum dos casos, é determinar que o software terá apenas um tipo de usuário. Dessa forma, podemos perder detalhes e construir um software que não atenda a todas as necessidades dos usuários.

Vamos utilizar um exemplo de um site de classificados de veículos. Será que lá teremos apenas um tipo de usuário? Vamos usar alguns nomes hipotéticos para descrever a situação.

- Alfredo é um consultor de vendas independente e atua em uma grande capital. Seus clientes têm interesse em adquirir veículos seminovos *premium* com qualidade. Muitos procuram veículos exclusivos e blindados. A maior parte das transações é a vista ou utilizando financiamento através dos bancos que o cliente já tem relacionamento. O próprio Alfredo busca veículos para seus clientes.
- Fred é consultor de negócio em uma loja de tamanho médio (Padrão Veículos) e atua em um bairro popular. Seu público é de pessoas que procuram seu primeiro veículo, na maioria das vezes utilizando financiamento com bancos parceiros da loja

com uma entrada entre 10 e 30% do valor do veículo. Para obter maior comissão no final do mês, Fred também busca oportunidades de compra para a loja, mantendo o estoque saudável.

- A Rede de Concessionárias Floresta é uma das maiores da região Sudeste. Tem várias filiais em diversas cidades nos estados de São Paulo, Rio de Janeiro e Minas Gerais. Comercializa veículos novos de duas montadoras e usados multimarcas. Em suas 30 lojas, tem em média 5 vendedores em cada, 1 avaliador e 1 captador de veículos.
- Pedro tem uma empresa de importação e busca comprar um veículo novo, já que seu anterior foi indenizado integralmente pela seguradora devido a um acidente. Procura um SUV seminovo, e tem disponibilidade para pagar à vista, até R$ 200k
- Maria é uma jovem estudante universitária. Sua mãe decidiu comprar um carro para ela, dispõe de cerca de R$ 15k, ao buscar no mercado puderam se certificar que com esse valor conseguem comprar apenas veículos mais antigos. Estão dispostas a dar este valor de entrada se encontrarem um bom veículo mais novo com um financiamento acessível. A mãe de Maria também tem um carro que pode ser vendido para ajudar na entrada.
- João compra e vende carros como um negócio próprio. Ele foca em veículos abaixo da tabela, com pequenos problemas em que ele percebe que o vendedor aceita uma proposta mais ousada. Após a compra João dá um "trato" nos veículos e frequentemente consegue vende-los pelo preço de tabela ou até um pouco mais.

Veja que, nesse conjunto de tipos de usuário, pode-se identificar vários padrões diferenciados. A sugestão de Cohn (2004) é fazer uma discussão em cima do conjunto inicial de funções, organizar e consolidar funções e, depois, refiná-las.

Quadro 1 – Perfis identificados

| **Tipo de Função** | **Tipo específico** | **Enquadramento** |
|---|---|---|
| Comprador | Loja | Rede de Concessionárias Floresta<br>Padrão Veículos |
| | Pessoa Física<br>(para uso pessoal) | Pedro<br>Maria<br>Clientes do Alfredo<br>Clientes da Rede de Concessionárias Floresta<br>Clientes da Padrão Veículos |
| | Pessoa Física<br>(Como negócio) | Alfredo – Consultor Independente<br>João – Negócio próprio<br>Fred<br>Captadores da Rede de Concessionárias Floresta |

| | | |
|---|---|---|
| Vendedor | Loja | Rede de Concessionárias Floresta<br>Padrão Veículos |
| | Pessoa Física<br>(para uso pessoal) | Maria |
| | Pessoa Física<br>(Como negócio) | Alfredo – Consultor Independente<br>João – Negócio próprio<br>Fred – Padrão Veículos |

Fonte: o autor

Observe que, caso tivéssemos parado o refinamento apenas com o tipo de função, teríamos apenas quatro padrões de usuário, o que não declararia completamente os perfis que se deseja atender com o sistema. Observe, também, que alguns tipos de usuários foram inclusos em mais de um perfil.

Isso é muito comum e representa que esses usuários são mais diversificados. Identificado isso, devemos partir para a coleta das histórias de usuário. Diferentemente da engenharia de software tradicional, em que os usuários são entrevistados e fornecem informações apenas no momento inicial do processo de desenvolvimento, quando se utiliza histórias de usuário, esse é só um passo inicial.

Outro ponto importantíssimo é incentivar que os próprios usuários escrevam as histórias, pois elas devem ter um jargão de negócios, não técnico. No momento inicial de um projeto, devem ser realizadas sessões de escritas de histórias com o intuito de conhecer o escopo principal da solução desejada.

É compreensível que muitos usuários possam sentir resistência em começar a escrever histórias de usuário. Esse sentimento inicial é uma reação normal, e para superá-lo, é importante apresentar o método de forma clara e motivadora. É importante destacar que a escrita do cartão de história é apenas o primeiro passo no processo. O cartão é utilizado como uma referência para o assunto da história, mas ao longo da conversa sobre o tema, os detalhes serão aprofundados e, assim, será possível escrever os critérios de aceitação de maneira mais completa e precisa.

A escrita dos critérios de aceite pode ser abordada da seguinte forma com o cliente:

- O que você validaria ao testar essa história se ela estivesse pronta agora?
- Que regras você verificaria se ela atende?
- Que tipo de validações você faria?
- Existe alguma restrição ou limitação que você gostaria de destacar?

Pode parecer desafiante para usuários "comuns" no início, mas é importante lembrar que a prática leva à perfeição. Conforme os usuários vão ganhando experiência em definir critérios de aceitação, ficam mais habilidosos em aplicá-los de forma efetiva.

Um aspecto importante a ser levado em conta, é que a maioria dos projetos de software terão muitas histórias de usuário. Para isso, o conceito de épicos de histórias de usuário pode ser utilizado. Uma definição de épico foi apresentada por Cohn (2004), em que ele descreve um épico como "uma grande história de usuário que é composta de várias histórias menores". Ele também explica que um épico pode ser definido como uma história de usuário que é grande demais para caber em uma única iteração ou sprint e que, portanto, precisa ser dividida em histórias menores e mais gerenciáveis.

Os épicos de histórias de usuário são uma forma de agrupar um conjunto de histórias de usuário relacionadas em um único pacote. Um épico de história de usuário é uma história de alto nível, que descreve um conjunto de funcionalidades que um usuário deseja. O objetivo de um épico é fornecer uma visão geral dos recursos que um usuário necessita sem entrar em detalhes específicos, pois estes serão abordados nas histórias que compõe o épico.

Eles são geralmente escritos no início do projeto, quando a equipe de desenvolvimento ainda está trabalhando na definição do escopo do produto. Com estes macrorrequisitos, um entendimento da abrangência daquele sistema pode ser conhecido de forma preliminar, auxiliando na definição de viabilidade daquele projeto.

Os épicos ajudam a garantir que todas as histórias de usuário que possuam alguma relação estejam agrupadas juntas, o que pode simplificar a gestão do escopo e do desenvolvimento daquele produto.

Um dos benefícios mais significativos dos épicos de histórias de usuário é que eles permitem que a equipe de desenvolvimento priorize o trabalho. Ao agrupar as histórias de usuário em épicos, a equipe pode priorizar o trabalho com base na importância de cada épico e de cada história dentro do épico. Isso permite que a equipe se concentre nos épicos mais importantes primeiro e nas histórias mais importantes de cada épico.

Para representar como funciona uma especificação de um sistema dividida em épicos, apresentamos um exemplo de uma estrutura de épicos e histórias de usuário de um sistema para um laboratório de análises clínicas.

**Épico 1: Gerenciamento de cadastro de pacientes**

História de usuário 1.1: Cadastrar novo paciente

Como um funcionário do laboratório, eu quero ser capaz de cadastrar um novo paciente no sistema, para que possamos manter um registro de todos os pacientes que utilizam nossos serviços.

Critérios de aceitação:

O usuário deve ser capaz de preencher informações básicas do paciente, como nome, data de nascimento e gênero.

O sistema deve verificar se já existe um paciente cadastrado com as mesmas informações, e alertar o usuário se houver duplicidade.

O sistema deve gerar um número de identificação único para cada novo paciente cadastrado.

História de usuário 1.2: Editar informações de paciente

Como um funcionário do laboratório, eu quero ser capaz de editar as informações de um paciente já cadastrado no sistema, para que possamos manter os dados atualizados.

Critérios de aceitação:

O usuário deve ser capaz de visualizar todas as informações cadastradas do paciente.

O usuário deve ser capaz de editar as informações, exceto o número de identificação único.

O sistema deve registrar um log de alterações no cadastro do paciente.

História de usuário 1.3: Excluir paciente

Como um funcionário do laboratório, eu quero ser capaz de excluir um paciente do sistema, caso ele não utilize mais nossos serviços ou tenha falecido.

Critérios de aceitação:

O usuário deve ser capaz de buscar um paciente pelo número de identificação único ou pelo nome.

O sistema deve verificar se existem exames associados ao paciente a ser excluído, e alertar o usuário se houver.

O sistema deve permitir que o usuário escolha se deseja ou não manter um registro histórico do paciente excluído.

**Épico 2: Gerenciamento de Resultados de Análises Clínicas**

Descrição: Este épico trata da necessidade de gerenciar os resultados de análises clínicas realizadas pelo laboratório, permitindo a visualização, impressão e compartilhamento dos laudos pelos médicos e pacientes.

História de usuário 2.1: Visualização de resultados de análises

Como um médico ou paciente, eu quero visualizar os resultados das análises clínicas realizadas, para poder analisar e tomar decisões com base nos resultados.

Critérios de aceite:

Os resultados das análises devem ser apresentados de forma clara e organizada.

Os dados apresentados devem estar precisos e atualizados.

O acesso aos resultados deve ser seguro e restrito aos usuários autorizados.

História de usuário 2: Impressão de resultados de análises

História de usuário 2.2: Impressão de Resultados

Como um médico ou paciente, eu quero imprimir os resultados das análises clínicas realizadas, para ter uma cópia física dos resultados.

Critérios de aceite:

A impressão deve ser clara e legível.

Os dados impressos devem estar precisos e atualizados.

A impressão deve seguir um layout padrão para facilitar a compreensão.

História de usuário 2.3: Compartilhamento de resultados de análises

Como um médico ou paciente, eu quero compartilhar os resultados das análises clínicas realizadas, para poder enviar para outros profissionais ou compartilhar com familiares.

Critérios de aceite:

O compartilhamento deve ser feito de forma segura e restrita aos usuários autorizados.

Os dados compartilhados devem estar precisos e atualizados.

Deve ser possível escolher a forma de compartilhamento, como e-mail ou mensagem instantânea.

**Épico 3: Gerenciamento de laudos**

História de usuário 3.1: Visualização de laudos emitidos

Descrição

Como um médico responsável por acompanhar o tratamento de um paciente, eu quero visualizar todos os laudos emitidos para o paciente em questão, de forma a ter acesso rápido e fácil às informações relevantes.

Critérios de aceite

A busca pelos laudos deve ser feita pelo nome completo ou número de identificação do paciente.

A lista de laudos deve ser exibida em ordem cronológica decrescente.

Cada laudo deve exibir o nome do paciente, a data em que foi emitido e um resumo das informações mais relevantes.

História de usuário 3.2: Adição de anotações aos laudos

Descrição

Como um médico responsável por acompanhar o tratamento de um paciente, eu quero poder adicionar anotações aos laudos emitidos, de forma a incluir informações importantes para a compreensão do caso.

Critérios de aceite

A adição de anotações deve estar disponível apenas para usuários com permissão de edição nos laudos.

As anotações devem ser incluídas em um campo específico no laudo, que permita distinguir facilmente as informações adicionais das informações originais.

As anotações devem ser salvas imediatamente após a edição, e visíveis apenas para os usuários com permissão de edição.

História de usuário 3.3: Envio de laudos por e-mail

Descrição

Como um médico responsável por acompanhar o tratamento de um paciente, eu quero poder enviar laudos por e-mail para outros profissionais de saúde envolvidos no caso, de forma a compartilhar as informações relevantes.

Critérios de aceite

O envio de laudos por e-mail deve ser disponibilizado apenas para usuários com permissão de compartilhamento de laudos.

O e-mail deve conter um link para download do laudo em formato PDF, além do nome e informações de contato do paciente.

O conteúdo do e-mail deve ser personalizável pelo remetente, de forma a incluir informações adicionais, se necessário.

As histórias de usuário, como um artefato de documentação do software, devem ser estruturadas e armazenadas de acordo com as necessidades de cada projeto. Existem várias ferramentas disponíveis para documentar histórias de usuário, desde opções simples e gratuitas até soluções mais complexas e pagas. As ferramentas mais populares incluem:

Papel e caneta: uma maneira simples e acessível de documentar histórias de usuário é escrevê-las em papel e, em seguida, armazená-las em um local físico ou digital. Isso pode ser feito de forma individual ou em equipe e é útil para projetos menores ou para aqueles que estão apenas começando, ou será utilizado apenas para sessões de levantamento de histórias e depois transformado em algum outro meio de armazenamento. Ter estes documentos físicos, gera alguns tipos de problemas como: dificuldade de compartilhamento, dificuldade de alteração, dificuldade de na busca de informações.

- Planilhas: o uso de planilhas é uma maneira popular de documentar histórias de usuário. Ferramentas como o Microsoft Excel ou o Google Sheets podem ser usadas para criar uma lista de histórias de usuário e seus respectivos critérios de aceitação. Isso permite que as equipes classifiquem e priorizem as histórias, além de registrar o progresso do desenvolvimento. Tem como vantagens a facilidade de compartilhamento de informações e a simplicidade do seu uso, pois todos utilizamos planilhas em nossas atividades profissionais.
- Software de gerenciamento de projetos: O mercado oferece várias ferramentas de gerenciamento de projetos, como o Trello, Jira e Asana, que incluem recursos para documentar e rastrear histórias de usuário. Essas ferramentas permitem que as equipes criem cartões ou tickets para cada história de usuário e acompanhem o progresso em tempo real. O legal destas ferramentas é que permitem também gerenciar o desenvolvimento destas histórias.
- Ferramentas especializadas: Existem também ferramentas especializadas projetadas especificamente para documentar histórias de usuário, como o Miro, ProdPad e UserVoice. Essas ferramentas incluem recursos avançados para gerenciamento de produto, como a criação de roadmaps e a colaboração em equipe.

Incluir histórias de usuário no processo de desenvolvimento de software é um desafio e deve ser planejada. A implantação bem-sucedida de qualquer framework, metodologia ou processo requer uma abordagem prática e avaliada. É importante aplicar o processo e avaliar seus resultados em intervalos regulares. Se necessário, realizar ajustes para melhorar a eficiência e eficácia do processo. Além disso, é fundamental que as equipes continuem inspecionando e replanejando regularmente para garantir o crescimento e a evolução contínuos da equipe.

Esse é o princípio da melhoria contínua, que permite que as equipes evoluam de forma sustentável e eficaz. É importante manter esse espírito de constância e busca por melhorias para alcançar sucesso e resultados positivos na implantação de processos e metodologias.

Neste tema vimos que a especificação de requisitos e as histórias de usuário são partes importantes do processo de desenvolvimento de software. A especificação de requisitos é uma técnica que permite definir e documentar os requisitos funcionais e não funcionais de um sistema e outros aspectos essenciais para compreender o contexto daquele software. Já as histórias de usuário são uma técnica ágil que permite descrever as funcionalidades do sistema do ponto de vista do usuário final. Estas técnicas podem ser utilizadas individualmente, ou combinadas entre si, de acordo com o contexto da equipe e do projeto do software a ser desenvolvido.

# RESUMO DO TÓPICO 2

**Neste tópico, você aprendeu:**

- A importância da escrita de uma especificação de requisitos detalhada para garantir que todos os envolvidos no projeto estejam alinhados com as necessidades e expectativas do software a ser desenvolvido.

- Os aspectos importantes a serem considerados como critérios de qualidade na hora de preparar uma especificação de requisitos.

- Como elaborar histórias de usuário envolvendo as partes interessadas e utilizando o conceito INVEST para criar uma documentação adequada.

- Como construir critérios de aceite detalhados para histórias de usuário, utilizando linguagem natural ou a técnica de BDD – Behavior Driven Development.

# AUTOATIVIDADE

1 O acrônimo em inglês INVEST recomenda que uma história, para ser considerada excelente, deve atender a alguns requisitos. Qual é o objetivo do conceito INVEST no contexto de histórias de usuário e como ele pode ser aplicado para garantir que as histórias sejam eficazes e valiosas para a equipe de desenvolvimento e os usuários finais?

a) ( ) O objetivo do INVEST é definir um processo de planejamento para as histórias de usuário, garantindo que elas sejam implementáveis e testáveis.
b) ( ) O objetivo do INVEST é fornecer uma estrutura para desenvolver histórias de usuário que são independentes, negociáveis, valiosas, estimáveis, pequenas e testáveis.
c) ( ) O objetivo do INVEST é garantir que as histórias de usuário sejam escritas em linguagem clara e acessível, para que todos os membros da equipe possam entendê-las.
d) ( ) O objetivo do INVEST é acelerar o processo de desenvolvimento, permitindo que as histórias de usuário sejam entregues mais rapidamente.

2 A especificação detalhada de requisitos é uma etapa crucial na concepção de um sistema, pois permite que as expectativas e necessidades do usuário sejam claramente definidas e entendidas pelo time de desenvolvimento. Qual é a importância da especificação detalhada de requisitos no processo de desenvolvimento de software e como ela pode ser utilizada para garantir o sucesso do projeto?

a) ( ) A especificação detalhada de requisitos ajuda a garantir que os desenvolvedores criem o software correto, mas não tem impacto no sucesso geral do projeto.
b) ( ) A especificação detalhada de requisitos não é importante, pois os desenvolvedores devem ser capazes de entender e implementar as necessidades dos usuários sem a necessidade de documentação.
c) ( ) A especificação detalhada de requisitos é fundamental para garantir que o software atenda às necessidades dos usuários e ajuda a evitar atrasos e custos adicionais no projeto.
d) ( ) A especificação detalhada de requisitos é importante, mas apenas para projetos grandes e complexos, e pode ser ignorada em projetos menores e mais simples.

3 A especificação detalhada deve incluir tanto os requisitos funcionais quanto os requisitos não funcionais do sistema. Quais são as consequências de falhas no processo de detalhamento de requisitos de um software?

a) ( ) Falhas no processo de detalhamento de requisitos podem levar a atrasos no desenvolvimento do software, mas não afetam a qualidade final do produto.

b) ( ) Falhas no processo de detalhamento de requisitos podem levar a custos adicionais, retrabalho e possíveis erros no software final.
c) ( ) Falhas no processo de detalhamento de requisitos não são importantes, pois os desenvolvedores podem usar sua intuição e conhecimento técnico para projetar o software.
d) ( ) Falhas no processo de detalhamento de requisitos afetam apenas a comunicação entre a equipe de desenvolvimento e o cliente, mas não afetam diretamente o software.

4 Imagine o seguinte cenário: Alberto é usuário de uma plataforma de seu banco, que dá *cashback* (retorna dinheiro) de acordo com compras que ele faz utilizando o cartão de crédito. Sobre o exposto, escreva uma história de usuário acerca do problema mostrado.

5 Uma especificação detalhada de requisitos deve incluir informações como objetivos de sistema requisitos funcionais, requisitos não funcionais, entre outros. Elabore uma especificação de requisitos detalhada referente a um software de aluguel de veículos. Inclua o objetivo do sistema, requisitos funcionais, não funcionais, tipos de usuários e restrições.

# UML – *UNIFIED MODEL LANGUAGE*

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 3, abordaremos a UML, ou *Unified Modeling Language*, que é uma linguagem de modelagem visual utilizada para representar sistemas de software. Ela é utilizada para descrever e documentar sistemas de software de forma clara e precisa, garantindo que todos os envolvidos na implementação tenham uma compreensão comum do projeto. Com o uso de UML, é possível visualizar e compreender de forma clara as relações e interações entre os componentes do sistema, bem como sua estrutura e comportamento.

UML é uma linguagem ampla e flexível, e permite que modelos sejam criados para representar uma ampla gama de sistemas, desde pequenas aplicações até sistemas complexos e distribuídos. Além disso, UML é uma linguagem padronizada, o que significa que é amplamente aceita e utilizada em todo o mundo. Dessa forma, permite que equipes distribuídas trabalhem juntas, com uma compreensão comum do projeto.

A utilização de UML é fundamental em projetos de software, pois permite a comunicação clara e efetiva entre todos os envolvidos, desde analistas de negócios até desenvolvedores. Além disso, ela facilita a identificação de problemas e a tomada de decisões, permitindo que as equipes de desenvolvimento de software trabalhem de forma mais eficiente e eficaz. Nesse tema de aprendizagem, abordaremos os conceitos básicos de UML, suas aplicações e a importância da sua utilização em projetos de software e focaremos esta abordagem nos modelos de caso de uso e de atividades.

## 2 UML – *UNIFIED MODEL LANGUAGE*

UML (*Unified Modeling Language*) é uma linguagem de modelagem de sistemas de software que fornece uma notação visual para representar e documentar as estruturas de sistemas de software (BOOCH; RUMBAUGH; JACOBSON, 1999). Ela permite que os desenvolvedores representem graficamente a arquitetura, comportamento e relações entre objetos em um sistema de software.

UML é amplamente utilizado em todo o mundo para desenvolvimento de software, pois permite aos desenvolvedores comunicarem ideias e conceitos de forma clara e eficiente (FOWLER, 2004). Ele também é útil na documentação de sistemas de software e na verificação de sua arquitetura antes da codificação começar (BLAHA; RUMBAUGH, 2006).

Sommerville (2018, p. 122), afirma que a "UML é um conjunto de 13 tipos diferentes de diagramas que podem ser utilizados para modelar sistemas de software". Este trabalho surgiu nos anos 1990, com a integração de uma série de notações que já existiam para modelar sistemas. Por isso o nome é *Unified Model Language*, que significa Linguagem de Modelagem Unificada. Em 2004, foi publicada uma importante revisão, denominada UML 2.

É importante ressaltar a relevância da UML para a modelagem de software. É um padrão aceito universalmente e que se aplica em todos os tipos de software. Para Sommerville (2018), os modelos podem ser utilizados para três aspectos:

- Gerar e focar a discussão sobre um sistema existente ou que está sendo proposto. Mesmo que sejam incompletos, mas que tenham cobertura sobre os pontos-chave da discussão irão permitir uma dinâmica de entendimento sobre o sistema em questão;
- Para que se possa documentar um sistema existente. Podem, inclusive, ser utilizados para documentar apenas partes mais importantes de um sistema, obviamente utilizando a notação de forma correta;
- Como parte do processo de construção de um software, sendo utilizado como base para as etapas de implementação dele. Neste contexto, devem ser completos e corretos, pois o código-fonte e o comportamento do sistema, serão construídos a partir destes diagramas.

Ao observar que são 13 tipos diferentes de diagramas, pode-se gerar uma expectativa negativa. Afinal, parece bastante trabalho para elaborar todos estes diagramas para um software. A UML não obriga que se utilize todos os seus diagramas, conhecendo-os, pode-se aplicar um conjunto de diagramas mais adequado para o contexto do software em questão. Obviamente alguns diagramas são mais genéricos, se enquadram em qualquer software, outros vai depender do tipo de complexidade que o software possui.

De acordo com um estudo realizado por Erickson e Siau (2007 *apud* SOMMERVILLE, 2018), a maioria dos usuários da UML acreditava que cinco tipos de diagramas eram suficientes para representar os principais aspectos de um sistema. Esses diagramas incluem:

- Diagramas de atividades, que retratam as etapas envolvidas em um processo ou no processamento de dados.
- Diagramas de casos de uso, que ilustram as interações entre um sistema e seu ambiente.
- Diagramas de sequência, que mostram as interações entre os atores e o sistema, bem como entre os componentes do sistema.
- Diagramas de classes, que apresentam as classes de objetos no sistema e as relações entre elas.
- Diagramas de máquinas de estados, que exibem como o sistema responde a eventos internos e externos.

Esses cinco tipos de diagramas são considerados fundamentais para a modelagem de sistemas utilizando a UML e são amplamente utilizados pela comunidade de desenvolvimento de software.

# 3 DIAGRAMAS DE CASO DE USO

Dentro da UML, um dos diagramas que representam a interação do sistema é o Diagrama de Casos de Uso. A concepção desta técnica foi originalmente desenvolvida por Ivar Jacobson nos anos 1990.

Basicamente, um caso de uso é uma descrição simples do que um usuário espera de um sistema naquela interação. Cada caso de uso trata especificamente de uma tarefa que envolve uma interação externa com um sistema (SOMMERVILLE, 2018).

Um caso de uso é uma representação detalhada das interações entre o sistema em questão e seus atores externos, visando alcançar um objetivo específico. Além disso, é importante destacar que o sistema possui responsabilidades perante as partes interessadas, e essas responsabilidades precisam ser capturadas na definição dos requisitos do sistema (COCKBURN, 2005).

Um ponto importante ressaltado por Van Lamsweerde (2009) é que os casos de uso e os requisitos funcionais são duas abordagens diferentes na documentação de requisitos de software. Os requisitos funcionais as necessidades que o software deve atender, enquanto os casos de uso descrevem como o software deve funcionar para atender estas necessidades. Em outras palavras, os requisitos funcionais especificam as funcionalidades do software, enquanto os casos de uso descrevem os cenários de uso do software.

Os requisitos funcionais são geralmente escritos na forma de declarações curtas e objetivas, que descrevem as funcionalidades do software. Eles são usados para definir as funcionalidades do software que são necessárias para atender às necessidades do usuário. Por outro lado, os casos de uso são histórias que descrevem as interações entre o usuário e o sistema, ajudando a entender como o software será usado na prática (VAN LAMSWEERDE, 2009).

Por exemplo, imagine que está sendo desenvolvido um sistema de compras on-line para uma loja de eletrônicos. Um caso de uso para esse sistema poderia ser "Efetuar compra", no qual seriam descritas as interações entre o usuário, o sistema e o processo de pagamento para que o usuário possa realizar uma compra com sucesso. Além disso, o sistema teria a responsabilidade de garantir a segurança dos dados do usuário durante o processo de compra, o que deve ser considerado na definição dos requisitos do sistema.

Complementando este conceito, Cockburn (2005) afirma que um caso de uso captura um contrato, que descreve o comportamento do sistema sob várias condições, à medida que o sistema responde a uma solicitação de um de seus envolvidos. Um caso de uso representa uma jornada formatada sobre como um usuário interage com um sistema através de um conjunto de situações determinadas e específicas. Um caso de uso representa o software do ponto de vista do usuário.

Um aspecto importante a ser considerado é que casos de uso são instrumentos de comunicação dentro de um processo de desenvolvimento de software. O uso de casos de uso como ferramenta de comunicação é fundamental para garantir que a equipe de desenvolvimento esteja alinhada com as necessidades do usuário. Isso ocorre porque a descrição dos casos de uso é feita em linguagem natural e ilustrada com diagramas, o que torna mais fácil para os membros da equipe compreenderem o que deve ser implementado. Além disso, a descrição dos casos de uso permite que os usuários validem as funcionalidades antes da implementação, evitando problemas de usabilidade ou inconsistências com as expectativas do usuário.

No contexto da elaboração de casos de uso, o primeiro aspecto a ser definido, é o conjunto de atores que são envolvidos naquele contexto. Atores são representações de pessoas ou dispositivos que exercem papéis em relação à operação daquele sistema, seja enviando ou recebendo informações (PRESSMAN; MAXIM, 2016).

Em termos de desenvolvimento de software, um ator é uma entidade que possui comportamento. Isso pode incluir pessoas, empresas, programas de computador, sistemas de computador, hardware ou software, ou uma combinação desses elementos. Em vez de chamar de "ator", seria mais adequado se referir a esse conceito como "papel", ou seja, o papel desempenhado por uma pessoa ao utilizar o sistema. Por exemplo, esse papel pode ser "criador de fatura", "tomador de pedido" ou "gerente". No entanto, a palavra "ator" é amplamente utilizada na indústria de software e é compreendida adequadamente. Em vez de falar de "tipo de ator" ou "função", que indicam uma categoria de pessoas, como uma descrição de cargo ou relacionamento, é mais comum utilizar a palavra "ator" (COCKBURN, 2005).

É comum confundirmos usuário de um sistema, com ator do sistema. Em modelagem de casos de uso, um ator é uma entidade externa ao sistema que interage com ele, enquanto um usuário é uma pessoa que utiliza o sistema. Em outras palavras, um ator é um papel que pode ser desempenhado por uma ou mais pessoas, enquanto um usuário é uma pessoa específica.

Por exemplo, em um sistema de compras on-line, um ator pode ser um cliente e um usuário pode ser uma pessoa específica que está realizando compras. O ator cliente pode ser composto por vários usuários diferentes que utilizam o sistema. Em resumo, a diferença entre ator e usuário em casos de uso é que o primeiro representa um papel genérico e o segundo representa uma pessoa específica que ocupa esse papel.

A definição de atores é muito relevante para que o entendimento do sistema se torne o mais detalhado possível. No mesmo contexto de um sistema de compras on-line, supomos que exista clientes que são considerados clientes padrões, que não desfrutam de nenhum tipo de tratamento ou benefício diferenciado, e tenhamos os clientes fidelidade, que possuem benefícios e funcionalidades diferenciadas em algumas situações de interação com a loja virtual. Neste contexto, é importante definir dois atores para cliente. "Cliente Convencional" e "Cliente Fidelidade".

Em sua obra, Pressman e Maxim (2016) alertam para o fato que a identificação de atores é evolutiva tal qual a atividade de levantamento de requisitos. Na primeira interação com relação à definição do software, os atores primários serão identificados, e a cada nova interação, com novos fatores conhecidos sobre aquele sistema, os atores secundários serão identificados. O conceito é que os atores primários têm a responsabilidade de atingir a função necessária e obter o benefício desejado com aquele sistema e os secundários dão suporte para que os primários possam realizar o seu trabalho.

Algumas falhas podem ocorrer no processo de definição de atores de um software, das quais podemos destacar:

- Definição imprecisa: Definir um ator de forma vaga ou imprecisa pode prejudicar a compreensão do caso de uso e dificultar a implementação.
- Falta de identificação de atores: Não identificar todos os atores relevantes pode levar a uma compreensão incompleta do sistema e ao desenvolvimento inadequado do caso de uso.
- Atores duplicados: Definir múltiplos atores que exercem o mesmo papel pode levar a redundâncias e confusão.
- Atores irreais: Definir atores que não existem na realidade ou que não são relevantes para o sistema pode prejudicar a qualidade do modelo.
- Atores externos não identificados: Não identificar atores externos ao sistema, como fornecedores ou parceiros, pode prejudicar a compreensão das interações com o sistema.

Após definir os atores de um caso de uso, algumas perguntas podem ser respondidas, para que se possa estruturá-lo (PRESSMAN; MAXIM, 2016):

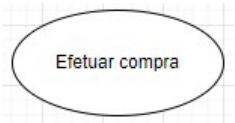
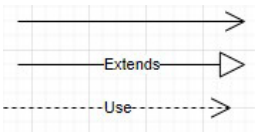
Quem é o ator primário e quem é (são) o(s) ator(es) secundário(s)?
Quais são as metas do ator?
Que precondições devem existir antes de uma jornada começar?
Que tarefas ou funções principais são realizadas pelo ator?
Que exceções poderiam ser consideradas à medida que uma jornada é descrita?
Quais são as variações possíveis na interação do ator?
Que informações de sistema o ator adquire, produz ou modifica?
O ator terá de informar o sistema sobre mudanças no ambiente externo?

Que informações o ator deseja do sistema?
O ator gostaria de ser informado sobre mudanças inesperadas?

Esta lista de perguntas pode ser sempre revisitada ao construir um caso de uso, à medida que a experiência com casos de uso vai aumentando, o analista passa a estruturar o seu caso de uso pensando nestes aspectos quase que automaticamente. Claro que pode-se criar um checklist após elaborar o caso de uso para verificar se todos os pontos citados foram considerados.

A notação do diagrama de caso de uso é bastante objetiva e envolve principalmente três elementos, descrita no Quadro 2:

Quadro 2 – Elementos da Notação de Casos de Uso

| Elemento da Notação | Descrição |
|---|---|
|  | Caso de uso: formato elipse na horizontal e que representam os diferentes usos que um usuário pode ter. |
|  | Atores: bonecos palito, representando os papéis que implementam os casos de uso. |
|  | Associações: uma linha entre atores e casos de uso. Estas associações determinam o tipo de relacionamento que há entre os atores e casos de uso e entre casos de uso. |

Fonte: adaptado de Pressman e Maxim (2016)

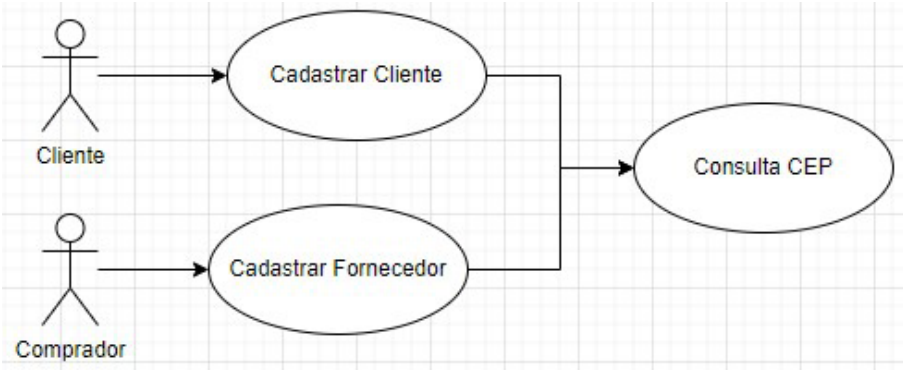
**IMPORTANTE**

Para desenhar um Diagrama de Casos de Uso, há uma série de ferramentas disponíveis, inclusive on-line, que disponibilizam os elementos da notação. Podemos citar por exemplo a ferramenta draw.io, que é gratuita e disponível em: http://draw.io.

No desenho de um diagrama de casos de uso, é possível demonstrar vários tipos de relacionamentos entre casos de uso. O relacionamento de inclusão ocorre quando existe um bloco de comportamento que é semelhante em mais de um caso de uso e pode-se facilitar o trabalho criando um caso de uso que poderá ser utilizado por ambos.

Por exemplo, em um sistema empresarial, existe a necessidade de através do CEP obter um endereço. Isso pode ser utilizado em vários casos de uso, como Cadastrar Cliente, Cadastrar Funcionário, Cadastrar Fornecedor, em qualquer caso de uso que precise de endereço. O diagrama, neste caso, fica da seguinte forma:

Figura 3 – Diagrama de Casos de Uso com relacionamento do Tipo Inclusão




Fonte: o autor

Um caso de uso é composto por um conjunto de cenários que são uma sequência de passos que descrevem uma interação entre um usuário e um sistema. Portanto, se tivermos uma loja on-line, podemos ter um cenário de compra de um produto que diria o seguinte:

- O cliente navega no catálogo e adiciona os itens desejados a cesta de compras. Quando o cliente deseja pagar, o cliente descreve as informações de entrega e cartão de crédito e confirma a venda.
- O sistema verifica a autorização do cartão de crédito e confirma a venda imediatamente e encaminha um e-mail de confirmação.

Este cenário é o caminho feliz, uma forma de essa sequência acontecer. No entanto, a autorização do cartão de crédito pode falhar. Este seria um cenário separado, muitas vezes chamado de cenário alternativo. Segundo Cockburn (2005), os cenários alternativos de casos de uso são usados para descrever as variações ou exceções que podem ocorrer quando o fluxo principal de eventos de um caso de uso é executado. Eles fornecem uma visão mais completa do comportamento do sistema e ajudam a identificar as condições em que o sistema pode falhar.

Um caso de uso é um conjunto de cenários vinculados por um usuário comum. Na situação anterior, você teria um caso de uso Comprar um Produto com a compra bem-sucedida e a falha na autorização como dois dos cenários do caso de uso. Outros caminhos alternativos para o caso de uso são mais cenários a serem considerados.

A quantidade de detalhes que você precisa depende do risco no caso de uso: quanto mais risco, mais detalhes você precisa. Muitas vezes, ficará claro que será necessário entrar em detalhes em alguns casos de uso durante a elaboração, e o restante não contém mais do que o caso de uso padrão.

Em complementação ao diagrama de casos de uso, pode-se escrever uma especificação do caso de uso. Há muita variação na forma como você pode descrever o conteúdo de um caso de uso, a UML não especifica nenhum padrão. Há também seções adicionais que podem ser adicionadas. Por exemplo, você pode adicionar uma seção para as pré-condições, que são aspectos que devem ser verdadeiras quando o caso de uso pode iniciar, ou pós-condições que determinam em que estado algo deve estar após a execução do caso de uso.

**Exemplos de especificação de caso de uso**

**Caso de uso**: UC01 - Realizar uma aposta

Ator principal: Apostador
Pré-condições: o usuário deve ter uma conta válida no aplicativo de apostas on-line e ter saldo suficiente em sua conta para realizar a aposta desejada.

Cenário principal: o usuário seleciona uma aposta em um evento esportivo disponível no aplicativo.

O usuário abre o aplicativo e seleciona a seção "Apostas".
O usuário seleciona o evento esportivo em que deseja realizar a aposta.
O usuário escolhe a opção de aposta desejada (por exemplo, "Vencedor do jogo").
O usuário digita o valor da aposta e confirma sua escolha.
O aplicativo verifica se o usuário tem saldo suficiente em sua conta para cobrir a aposta.
Se houver saldo suficiente, o aplicativo confirma a realização da aposta e o valor é debitado da conta do usuário.
O usuário é notificado da realização da aposta e pode visualizá-la em sua seção "Histórico de apostas".
Pós-condições: a aposta foi realizada com sucesso e o usuário pode acompanhar o resultado do evento esportivo em questão. Se a aposta for vencedora, o valor será creditado na conta do usuário.

**Cenário alternativo 1**: Saldo insuficiente na conta do usuário

O usuário segue os passos 1 a 4 descritos no cenário principal.
O aplicativo verifica se o usuário tem saldo suficiente em sua conta para cobrir a aposta.

Se não houver saldo suficiente, o aplicativo informa ao usuário que é necessário recarregar sua conta antes de realizar a aposta.

O usuário pode escolher recarregar sua conta ou cancelar a operação.

**Cenário alternativo 2**: Evento esportivo cancelado ou adiado

O usuário segue os passos 1 a 4 descritos no cenário principal.
Antes da confirmação da aposta, o evento esportivo é cancelado ou adiado.
O aplicativo informa ao usuário sobre o cancelamento ou adiamento do evento.
A aposta não é realizada e o valor não é debitado da conta do usuário.

**Caso de uso:** UC02 - Agendar Consulta Médica

Ator principal: Paciente

Pré-condições:
- O paciente deve ter uma conta no aplicativo.
- O paciente deve escolher uma data e hora futura.

Pós-condições:
- A consulta é registrada no sistema e confirmada para o paciente por e-mail ou SMS.

Resumo: o paciente pode agendar uma consulta médica com um médico específico em um determinado horário.

Cenário principal
O paciente acessa o aplicativo de agendamento de consultas médicas.
2. O paciente seleciona o médico que deseja consultar.
3. O paciente escolhe uma data e hora disponível para a consulta.
4. O paciente confirma a consulta.
5. O aplicativo registra a consulta no sistema e envia uma confirmação por e-mail ou SMS para o paciente.

Fluxo alternativo:
2a. Se o médico selecionado não estiver disponível na data e hora escolhidas, o aplicativo informa ao paciente e volta para o passo 2.

Extensões:
- Caso o paciente deseje cancelar a consulta, ele pode fazê-lo através do aplicativo.
- O aplicativo pode enviar lembretes por e-mail ou SMS para o paciente antes da consulta.

Os casos de uso e os diagramas de caso de uso são importantes ferramentas no processo de desenvolvimento de software, permitindo que os desenvolvedores possam entender as necessidades do usuário e implementá-las de forma mais clara e eficiente. Eles ajudam a descrever as interações entre os usuários e o sistema, permitindo que os desenvolvedores possam ter uma visão mais completa do software a ser construído.

# 4 DIAGRAMAS DE ATIVIDADE

Diagramas de Atividade UML (Unified Modeling Language) são representações gráficas que descrevem o comportamento de um sistema, desde a inicialização até o término de uma tarefa. Eles ilustram as atividades que são realizadas pelos usuários ou sistemas dentro de um processo, bem como as relações entre essas atividades (PRESMANN; MAXIM, 2016).

A utilização dos diagramas de atividade UML é muito útil na modelagem e especificação de requisitos de software. Eles permitem que os desenvolvedores visualizem o comportamento do sistema de uma forma clara e concisa, facilitando a compreensão e comunicação dos requisitos com outros membros da equipe de desenvolvimento de software (SOMMERVILLE, 2018).

Os diagramas de atividade em UML incluem elementos como objetos, atividades, transições, fluxos de controle, estados e guardas. Esses elementos são usados para representar o processo, as ações e os fluxos de trabalho do sistema.

Ao criar um diagrama de atividade, é importante considerar o contexto e a finalidade do sistema. É necessário identificar todas as atividades envolvidas no processo e estabelecer relações entre elas, para garantir a compreensão completa do comportamento do sistema. É importante também considerar a ordem em que as atividades ocorrem e as condições que levam a mudanças de uma atividade para outra.

O diagrama de atividades é uma ferramenta muito adequada para ilustrar processos de negócios, pois permite descrever explicitamente eventos paralelos que ocorrem durante o processo. Diferentemente de processos lineares, os processos de negócios frequentemente apresentam paralelismos, o que pode ser facilmente representado através do uso do diagrama de atividades (GRÄSSLE; BAUMANN; BAUMANN, 2005).

Os diagramas de atividades podem ser desenvolvidos em diversos níveis de detalhe, permitindo refinamentos passo a passo. No entanto, é importante ressaltar que, assim como os diagramas de casos de uso, os diagramas de atividades representam apenas processos de negócios e atividades de uma perspectiva externa. Ou seja, a elaboração de diagramas mais detalhados não deve implicar na descrição de processos que ocorrem exclusivamente dentro do sistema de negócio, o que pode levar a uma visão interna e prejudicar a compreensão do processo de negócio (GRÄSSLE; BAUMANN; BAUMANN, 2005).

Os diagramas de atividade podem ser utilizados em várias fases do desenvolvimento de software, desde a especificação dos requisitos até a validação do projeto. Durante a especificação de requisitos, eles ajudam a identificar e visualizar as atividades que precisam ser realizadas pelo sistema, e a definir as relações entre elas. Já durante a validação do projeto, os diagramas de atividade UML podem ser usados para verificar se o sistema está comportando-se de acordo com as especificações.

O diagrama de atividades descreve o sequenciamento de atividades, com suporte para comportamento condicional e paralelo. Um diagrama de atividades é uma variante de um diagrama de estado em que a maioria, se não todos, os estados são estados de atividade. Assim, grande parte da terminologia segue a dos diagramas de estado.

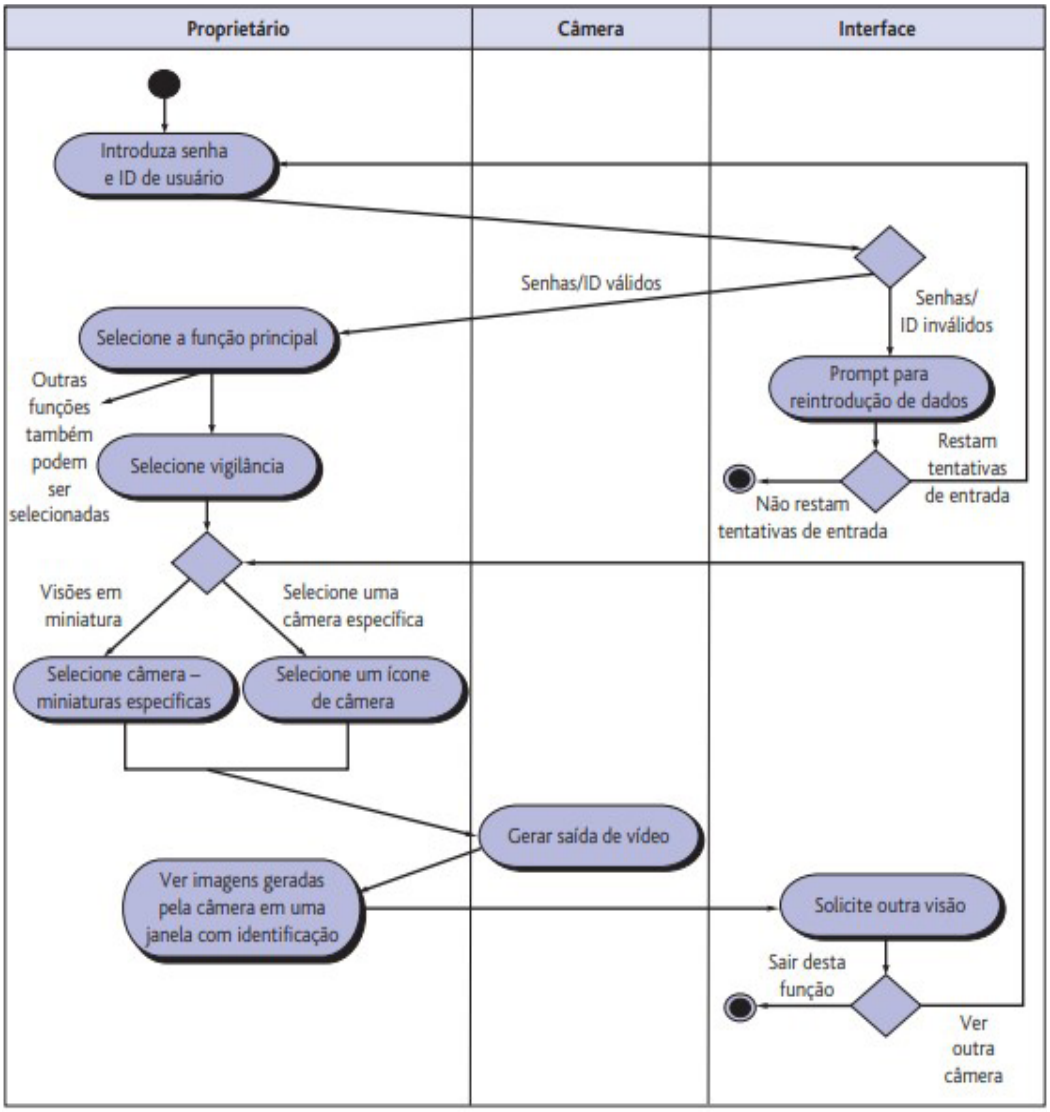
De acordo com Grässle, Baumann e Baumann (2005), os diagramas de atividades são compostos por diversos componentes fundamentais que ajudam a delimitar e compreender o processo representado. Esses componentes incluem a fase inicial e fase final, que marcam o início e o fim do processo, respectivamente. Além disso, há as atividades ou estados de ação, que representam as etapas do processo e colocam uma série de ações em movimento. As ações, por sua vez, são as tarefas executadas pelo sistema ou usuário, como verificar o saldo de uma conta bancária.

Os objetos também são importantes componentes dos diagramas de atividades, representando os materiais ou dados que são criados ou usados dentro de uma atividade. Já as decisões são as bifurcações do fluxo que exigem uma resposta positiva ou negativa antes de prosseguir para a próxima ação ou atividade. A sincronização é representada por meio dos nós de bifurcação e junção, marcando a criação e mesclagem de fluxos simultâneos. Os sinais, por sua vez, são usados para indicar como as ações podem ser realizadas fora do sistema e ainda assim alterar o processo, como no caso de uma integração para autorização de pagamento. Por fim, as swimlanes, ou raias, são as colunas ou categorias utilizadas para agrupar atividades relacionadas que são realizadas por diferentes atores.

Todos esses componentes são importantes para que o diagrama de atividades seja uma ferramenta efetiva de comunicação. Por meio deles, é possível visualizar o fluxo de atividades e as interações entre diferentes componentes e atores envolvidos no processo, tornando mais fácil a compreensão do processo e a identificação de possíveis problemas ou gargalos. Assim, os diagramas de atividades se mostram essenciais para a modelagem de processos e sistemas, auxiliando na comunicação entre os envolvidos e na tomada de decisões mais assertivas.

Na figura 4, pode-se observar um diagrama de atividades que representa o fluxo de um sistema de vigilância.

Figura 4 – Diagrama de Atividades




Fonte: adaptado de Sommerville (2018)

O diagrama de atividades é uma ferramenta poderosa para a modelagem e ilustração de processos de negócios, permitindo uma representação clara e detalhada dos eventos paralelos que podem ocorrer durante o processo. Por meio do uso de diagramas de atividades, é possível documentar os processos de negócios em diferentes níveis de detalhe, proporcionando uma visão geral do processo e permitindo refinamentos progressivos, assim completando a documentação do software de uma forma adequada.

Neste tema de aprendizagem, foi possível compreender que a UML é uma ferramenta importante para modelagem de sistemas, permitindo uma comunicação mais clara e precisa entre as equipes envolvidas no processo de desenvolvimento. Os

diagramas de caso de uso e de atividades são dois dos principais tipos de diagramas utilizados na UML e, quando combinados, podem fornecer uma visão completa e detalhada do sistema.

O diagrama de caso de uso é uma ferramenta eficaz para descrever os requisitos do sistema do ponto de vista do usuário, identificando as principais funcionalidades do sistema e as interações entre os usuários e o sistema, os cenários. Já o diagrama de atividades permite modelar o fluxo de trabalho do sistema, destacando as atividades envolvidas em cada etapa.

Quando combinados, esses dois tipos de diagramas permitem uma análise mais abrangente e detalhada do sistema, ajudando a identificar possíveis problemas ou limitações no fluxo de trabalho, bem como garantir que as funcionalidades do sistema atendam às necessidades dos usuários.

# LEITURA COMPLEMENTAR

**USER STORY MAPPING: TRANSFORME O BACKLOG DO PRODUTO EM MAPA**

Kellison Ferreira

User story mapping garante visualização otimizada e priorização adequada em backlogs de produto.

Se você está no início de sua carreira como product manager, ou está dentro de times de PM, certamente já se deparou com backlogs em que nada parecia muito claro. A infinidade de tasks gera dificuldade de visualizar o que é prioridade e como estruturar tarefas. É justamente por essa dificuldade que a técnica de user story mapping foi criada.

Processos ágeis devem ser também simples e com uma visualização facilitada. Do contrário, equipes perdem muito tempo e detalhes importantes sobre o mapeamento de pessoas usuárias podem ser negligenciados.

De nada adiantará um bom Product Discovery se as informações são complexas, massantes e pouco acessíveis quando chegam ao time de PM.

Neste post, nossa missão é mostrar que dá para fazer tudo diferente, e mais simples, com ajuda de user story mapping. Acompanhe para entender o que é, como funciona e as razões para adotar.

**USER STORY MAPPING: O QUE É?**

User story mapping é uma técnica de mapeamento que visa mostrar informações do backlog de produto de maneira descomplicada e visualmente acessível. Em resumo, todas as requisições são representadas, em nível hierárquico, por papéis adesivos (post-its) em um mural.

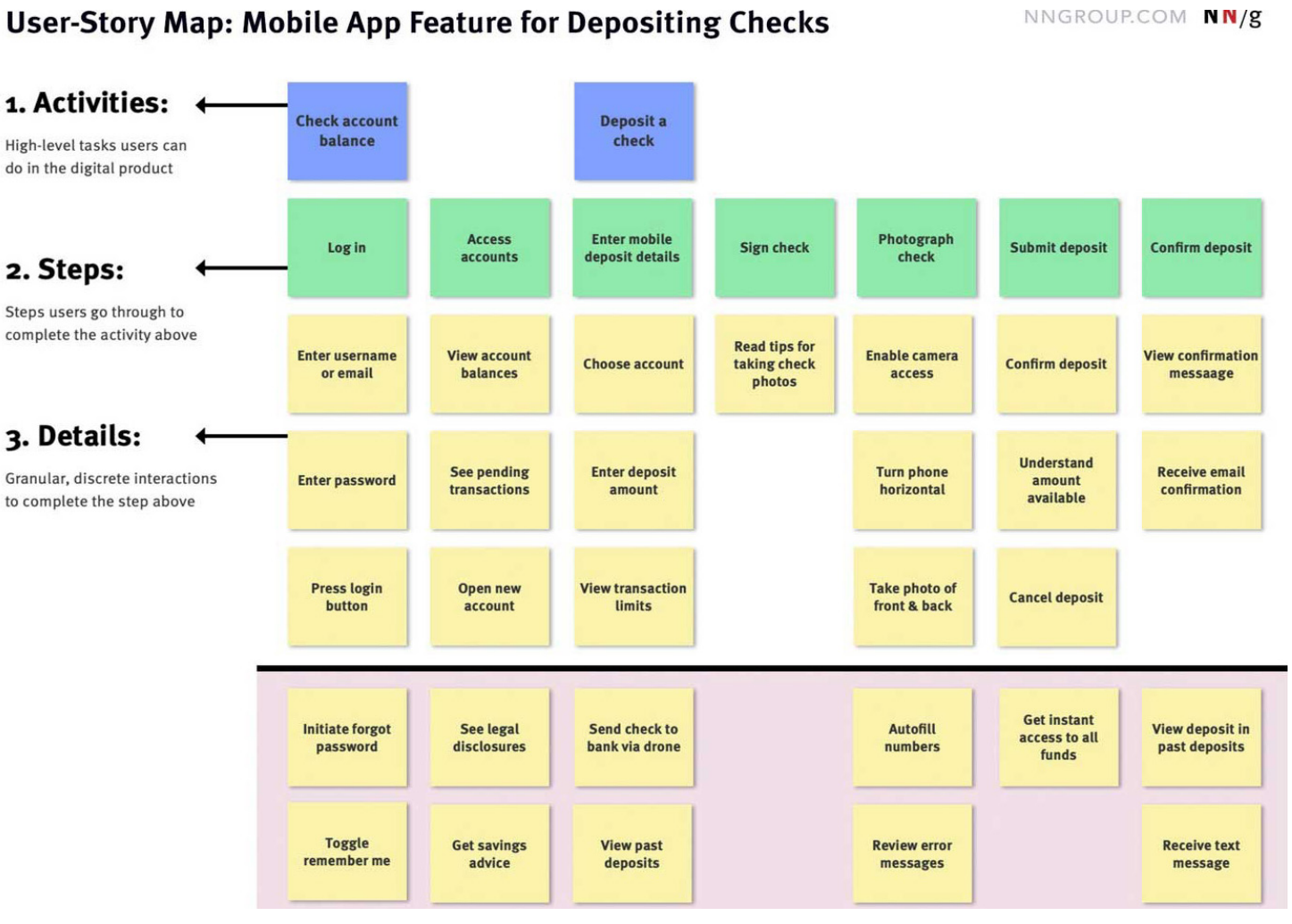
Parece simples, não? Na verdade, realmente é! Essa era a ideia de quem criou o processo, Jeff Patton, uma das referências mundiais em processos ágeis. Inclusive, você pode aprender um pouco mais com ele por meio de seu livro "User Story Mapping: Discover the Whole Story, Build the Right Product", além desta conferência, em que ele fala mais sobre o método e suas ideias.

O que motivou Jeff a criar essa mudança de abordagem foi justamente a alta complexidade dos métodos antigos. Times de produto precisam de dinamismo e acesso rápido às informações necessárias para o desenvolvimento do produto.

Se todas as percepções sobre exigências de pessoas usuárias ficam em um mural em que tudo é categorizado e escrito em frases curtas, a visualização se torna ágil e clara.

Sem esse método, times ficariam dependentes de longas documentações altamente aprofundadas e detalhadas sobre as requisições de UX. O problema é que isso toma tempo e exige muita energia para a absorção de todas as informações. Com story maps, se ganha em objetividade, comunicação, compreensão e dinâmica de trabalho. Além de, claro, assertividade na entrega de produtos.

Por mais que nossa explicação tente ser a mais objetiva e didática possível, você precisa realmente ver como esse mapeamento é feito para entender sua estrutura e funcionamento. Por isso, vamos mostrar um modelo utilizado pela companhia Norman Nielsen Group:




Fonte: Nielsen Norman Group

## NÍVEIS DE UM STORY MAP

Story maps se tornam realmente relevantes porque, além de dar suporte na visualização de todos os requisitos, também ajudam a priorizar tasks do backlog. Isso se faz por meio dos níveis, que você entende mais como funcionam e de que maneira direcionam o trabalho a seguir.

### Atividades

Podemos definir as atividades como as ações principais que pessoas usuárias desejam executar. Por exemplo, se elas acessam um app de delivery, essa atividade seria pedir uma refeição ou, se estamos falando de apps de deslocamento, seria solicitar um carro.

Conforme você pode conferir na imagem de exemplo da Norman Nielsen Group, essas atividades de alto nível encabeçam story maps. No produto em questão, todas as atividades precisam ser elencadas de maneira paralela, uma ao lado da outra. Isso ajuda a manter uma estruturação hierárquica, independentemente do número de ações de alto nível.

Como ações principais, podemos encarar essa primeira categoria como objetivos centrais, mas que demandam passos para serem alcançados, que é o que veremos em sequência.

### Passos

Logo abaixo das atividades vêm os passos, que retratam as etapas de ações menores necessárias para chegar à ação principal desejada por pessoas usuárias. Essas subtarefas vão compor uma jornada de atividades até completar objetivos principais dentro de produtos digitais dos mais variados tipos.

Se a ideia é solicitar um carro (atividade principal), passos necessários para isso são inserir a origem, ou seja, onde você está agora, e o destino, que é para onde deseja ir. Passos, geralmente, não se estendem tanto, já que são apenas o que vem antes da atividade.

### Detalhes

Por fim, o último nível de story maps são os detalhes. Na prática, são todas as pequenas ações necessárias para chegar ao último passo e, então, realizar atividades principais desejadas. São várias interações de menor impacto, mas que são recorrentes e devem ser consideradas para construir uma jornada de uso perfeita.

Aqui, é necessário pensar na experiência a partir do primeiro momento. Por exemplo, depois de abrir um app qualquer, pessoas usuárias precisarão clicar em botões de login, inserir suas credenciais e senha e então clicar em confirmar.

## BENEFÍCIOS DE MAPEAR USER STORIES

Mapear essa jornada de interações de pessoas usuárias é fundamental para processos de times ágeis. Há ganhos importantes em diversos pontos do trabalho de desenvolvimento. Por isso, vamos pontuar alguns dos principais a seguir. Acompanhe!

### Colaboração e alinhamento otimizados no time

Ter user story devidamente detalhado em painéis de post-its é uma forma eficaz de colocar processos em visualização ampla para times inteiros. Assim, ideias de qualidade podem ser debatidas de maneira simultânea. Além disso, pessoas do time de PM conseguem colaborar melhor umas com as outras e alinhar ações, já que a visualização é única.

### Backlogs com visualização mais simples

Backlogs em documentos complexos e grandes simplesmente não combinam com times ágeis. User story mapping pode ser considerada a solução ideal de colocar esses backlogs de maneira aberta, acessível e simplificada para cada pessoa da equipe. A estruturação em categorias e com ações e desmembramento usando papéis adesivos funcionam muito bem!

### Visualização facilitada de prioridades e Minimum Viable Product (MVP)

É fundamental pensar nas funcionalidades que são primordiais, ou seja, visualizá-las de maneira simples se faz necessário. User story mapping ajuda muito nisso, além de facilitar também a identificação de um MVP. Dessa forma, o tempo de desenvolvimento é otimizado e produtos podem ficar pronto para usos de teste, sendo minimamente funcionais.

### Identificação de elementos de risco facilitada

Boas ideias nem sempre vão se transformar em produtos funcionais. Por vezes, funcionalidades e etapas de ações não ajudam a construir a experiência perfeita. A parte boa de usar story maps é que esses elementos de risco podem ser identificados com facilidade nos murais. Uma vez que isso acontece, basta substituir as ações.

## MAPA DA HISTÓRIA DO USUÁRIO E MAPA DA JORNADA DO CONSUMIDOR

Os termos podem nos remeter às mesmas questões, mas a verdade é que eles são diferentes. Pessoas consumidoras e usuárias são as mesmas? Sim, mas a questão é a perspectiva de análise das atividades. São duas fases diferentes, o que faz toda a diferença.

Mapa da jornada de consumidor é uma análise dedicada a entender qual caminho essa pessoa faz durante a interação com determinado produto. A ideia é avaliar a jornada para realizar ações variadas e chegar a objetivos. São consideradas questões como as intenções, desejos e sentimentos dessas pessoas durante esse caminho.

Quando falamos de mapa da história do usuário a perspectiva é de análise de produto e de como deve funcionar para entregar a experiência perfeita. Então, caminhos são pensados para construir sequências de funcionalidades que geram interações simples de serem entendidas. Assim, pessoas usuárias conseguem fazer melhor uso de produtos.

A relação entre esses dois conceitos é simples: mapas de jornada de pessoas consumidoras servem como base para criar mapas da história de pessoas usuárias. Isso é possível porque é necessário, primeiramente, entender as intenções de quem usa produtos para, só então, desenvolver experiências de interações que atendam perfeitamente aos desejos das pessoas.

## COMO FAZER UM USER STORY MAP PASSO A PASSO

Hora de colocar essas ideias em prática! Criar uma estrutura de user story mapping do zero é importante para saber como usar a ferramenta. É isso que mostraremos como fazer na sequência. Veja quais são os cinco passos.

### 1. Escolha a ferramenta para organizar o mapa

Construir o mapa é uma parte importante do processo, mas que também demanda esforços, não podemos negar. Estruturar as ações e seus desdobramentos requer análise e, só então, o ato de elencar cada passo de pessoas usuárias. Fazer isso sem a ajuda de ferramentas de gestão de projetos pode tornar o trabalho mais complexo, demorado e passível de erros.

Miro + Asana é uma das principais opções de solução para essa tarefa, justamente porque oferece uma estrutura de quadro branco de operação muito simples. A partir disso, dá para criar story maps de maneira clara.

Jira é outra boa opção que deve ser considerada. A ferramenta, bastante conhecida por times de PM, é uma plataforma de gestão de tarefas em times ágeis, o que também permite estruturar as ações em telas de fácil compreensão.

### 2. Trabalhe em conjunto com a equipe

Antes de qualquer coisa, user story mapping é uma ferramenta colaborativa. Times inteiros têm acesso aos quadros e podem visualizar as tarefas e requisições de backlog de maneira simultânea. Naturalmente, para que essa estrutura de trabalho seja construída, é importante que a equipe toda esteja disposta a construir story mappings.

Percepções vindas a partir da análise do mapa de jornada do consumidor podem ser pontuadas por diversas pessoas. Isso criará uma visão mais ampla da situação e criará requisições diversas. Por mais que pareça muita informação, basta que haja um refinamento dessas questões, o que deixará o mapeamento mais enxuto.

Trabalho em equipe também traz percepções variadas, ou seja, pessoas em ocupações diferentes conseguem aplicar requisições baseadas em experiência e qualificação. A partir daí, é mais fácil ter MVPs precisos e com potencial de se tornarem produtos bons.

**3. Faça um mapa visualmente organizado**

O grande ponto de destaque nessa técnica é a visualização das tasks. Quanto mais acessíveis elas estiverem à primeira vista, melhor será o trabalho de times de desenvolvimento. A ideia de post-its é aplicada para garantir que todas as questões sejam vistas em uma visão simplificada, com desdobramentos hierarquizados.

Portanto, seguir os níveis de story map de maneira rigorosa se faz importante. Assim, pessoas integrantes dos times saberão quais são as ações principais e quais delas são parte da jornada de chegada aos objetivos. Portanto, posicionar hierarquicamente permite entendimento quase que imediato.

Para otimizar ainda mais essa compreensão, usar post-its de diferentes cores é uma sugestão pontual e que terá ótimo efeito. Para cada nível, uma tonalidade contrastante. Assim, dá para separar as categorias e trazer mais agilidade para compreensão e análise de backlog.

**4. Considere a visão da pessoa usuária, não do produto**

Montar um mapa da história do(a) usuário(a) requer uma orientação à experiência de uso de pessoas junto ao produto. Se você quer saber quais são os objetivos principais a serem cumpridos e as etapas que fazem parte dele, pense primeiramente na jornada de cada pessoa usuária.

Do momento em que acessa uma interface até conseguir a atividade principal, ou seja, o objetivo, essa pessoa usuária passará por etapas. A ideia é que esse ciclo de ações seja devidamente visualizado, se colocando no lugar de quem usará o produto. Essa é a melhor forma de construir uma experiência que funcione bem e seja dinâmica.

Um erro que afeta desenvolvimentos de qualidade é pensar no desenvolvimento de produtos focando apenas em sua interface e funcionamento. Se essa atividade não parte de quem vai usar, não é possível criar uma experiência de qualidade para pessoas usuárias.

**5. Faça ajustes, se necessário**

Ajustes sempre farão parte do processo. Mesmo que eles tomem tempo, gerem retrabalho ou interfiram na agilidade do trabalho de times de PM. É importante analisar por outra perspectiva: ajustes são oportunidades de entregar produtos realmente perfeitos a pessoas consumidoras. Afinal, esse é o objetivo de todo trabalho feito.

Com story maps, executar ajustes é muito mais fácil, graças à visualização dinâmica. Vamos supor que o problema em questão é entre a etapa de login de um app de banco digital e o momento em que pessoas usuárias acessam o extrato da conta.

Uma vez detectado esse ponto, basta intervir exatamente nessa brecha, fazendo a alteração diretamente entre os dois post-its que representam essas ações descritas. Isso garante um trabalho melhor direcionado e mais ágil.

Aplicar user story mapping no cotidiano de equipes de PM vai fazer muita diferença na agilidade de desenvolvimento e na priorização de tasks. Esse é o início de um caminho de maior produtividade, menos erros e, claro, produtos melhor orientados a pessoas usuárias.

Fonte: FERREIRA, K. **User story mapping: transforme o backlog de produto em mapa**. 2021. Disponível em: https://blog.somostera.com/product-management/user-story-mapping. Acesso em: 11 jan. 2023.

# RESUMO DO TÓPICO 3

**Neste tópico, você aprendeu:**

- A linguagem UML e sua origem.
- O formato e a linguagem do Diagrama de Caso de Uso.
- O contexto para criação de especificação de caso de uso.
- Os diagramas de atividades como uma forma de documentar o software.

# AUTOATIVIDADE

1 A utilização de UML é fundamental em projetos de software, pois permite a comunicação clara e efetiva entre todos os envolvidos, desde analistas de negócios até desenvolvedores. Sobre o UML, assinale a alternativa CORRETA:

a) ( ) UML é uma linguagem de programação utilizada para desenvolver aplicativos.
b) ( ) UML é uma ferramenta de gerenciamento de projetos.
c) ( ) UML é uma linguagem de modelagem visual utilizada para representar sistemas de software.
d) ( ) UML é um sistema operacional.

2 Dentro da UML, um dos diagramas que representam a interação do sistema é o Diagrama de Casos de Uso. A concepção desta técnica foi originalmente desenvolvida por Ivar Jacobson nos anos 1990. O que são casos de uso em UML?

a) ( ) Casos de uso são diagramas que representam a funcionalidade do sistema.
b) ( ) Casos de uso são modelos que representam a interação entre o sistema e os usuários.
c) ( ) Casos de uso são diagramas que representam a arquitetura do sistema.
d) ( ) Casos de uso são modelos que representam a estrutura de dados do sistema.

3 Ao criar um diagrama de atividade, é importante considerar o contexto e a finalidade do sistema. É necessário identificar todas as atividades envolvidas no processo e estabelecer relações entre elas, para garantir a compreensão completa do comportamento do sistema. Qual das seguintes opções apresenta um cenário adequado para utilizar o diagrama de atividades do UML?

a) ( ) Modelagem de redes de computadores em uma organização.
b) ( ) Criação de layouts de páginas web para um site de comércio eletrônico.
c) ( ) Representação de estruturas de dados em um sistema de gerenciamento de banco de dados.
d) ( ) Modelagem de fluxos de trabalho em processos de negócios de uma empresa.

4 Um caso de uso é uma representação detalhada das interações entre o sistema em questão e seus atores externos, visando alcançar um objetivo específico. Além disso, é importante destacar que o sistema possui responsabilidades perante as partes interessadas, e essas responsabilidades precisam ser capturadas na definição dos requisitos do sistema. Escreva o detalhamento de um caso de uso referente à publicação de posts em um blog.

5 Basicamente, um caso de uso é uma descrição simples do que um usuário espera de um sistema naquela interação. Cada caso de uso trata especificamente de uma tarefa que envolve uma interação externa com um sistema. Fale sobre os benefícios de se construir diagramas de caso de uso para um projeto de software

# REFERÊNCIAS


ADZIC, G. **Specification by Example: How Successful Teams Deliver the Right Software**. New York: Manning Publications, 2011.

BECK, K. *et al*. **Manifesto for agile software development**. 2001. Disponível em: http://agilemanifesto.org/. Acesso em: 1 fev. 2023

RUMBAUGH, J.; BLAHA, M. **Modelagem e Projetos Baseados em Objetos com UML 2**. 1ª Edição. Rio de Janeiro: Elsevier-Campus, 2006.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The unified modeling language user guide**. Reading, MA: Addison-Wesley, 1999.

CAMPOS, E. L. de. **Critérios de aceitação das user stories**. 2011. Disponível em: https://bit.ly/42AtR5O. Acesso em: 1° fev. 2023

COCKBURN, A.; **Escrevendo Casos de Uso Eficazes - Um Guia para Desenvolvedores de Software**. São Paulo: Bookman, 2005

COHN, M. **User stories applied: for agile software development**. Boston: Addison-Wesley, 2004.

NIELSEN, J. **Usability Engineering**. San Francisco, CA: Morgan Kaufmann Publishers, 1993.

FOWLER, M. **UML distilled: A brief guide to the standard object modeling language**. Boston: Addison-Wesley Professional. 2004.

FOWLER, M. UserStory. 2013. Disponível em: https://martinfowler.com/bliki/UserStory.html. Acesso em: 2 fev. 2023.

RÄSSLE, P.; BAUMANN, H.; BAUMANN, P. **UML 2.0 in Action**: a project-based tutorial. Birmingham: Packt, 2005.

IEEE. **Guide to the Software Engineering Body of Knowledge: SWEBOK® v. 3.0**. Los Alamitos, CA: IEEE Computer Society, 2014.

IEEE. **IEEE Recommended Practice for Software Requirements Specifications**. IEEE Std 830-1998. New York: IEEE, 1998.

IEEE. **IEEE Std 829-2017**: Standard for Software and System Test Documentation. New York: IEEE, 2017.

IIBA. **Um guia para o corpo de conhecimento de análise de negócios: Guia BABOK®*. versão 2.0***. Toronto: theiiba.org, 2011.

JEFFRIES, R. **Essential XP**: card, conversation, confirmation. 2001. Disponível em: https://bit.ly/3pcXb3R. Acesso em: 1 fev. 2023.

KERR, E. S. **Gerenciamento de Requisitos**. São Paulo: Pearson, 2015

PRESSMAN, R.; MAXIM, B. R. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: Amgh, 2016.

REHKOPF, M. **User stories with examples and a template**: user stories are development tasks often expressed as "persona + need + purpose". 2023. Disponível em: https://bit.ly/41iBPj0. Acesso em: 14 abr. 2023.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018

SUTHERLAND, J.; SCHWABER, K. **The Scrum Guide**: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org, 2020. Disponível em: https://www.scrum.org/resources/scrum-guide. Acesso em: 14 abr. 2023.

VAN LAMSWEERDE, A. **Requirements Engineering: From System Goals to UML Models to Software Specifications**. Chichester: Wiley, 2009.

VAZQUEZ, C. E.; SIMÕES, G. S. **Engenharia de Requisitos**: software orientado ao negócio. Rio de Janeiro: Brasport, 2016.

WAKE, B. **INVEST in good stories**: the series. 2017. Disponível em: https://xp123.com/bonus/xp123.com-INVEST-series.pdf. Acesso em: 1 fev. 2023

WIEGERS, K. E.; BEATTY, J. **Software Requirements**. 3rd Edition. Redmond: Microsoft Press, 2013.

# QUALIDADE E GERENCIAMENTO DE REQUISITOS DE SOFTWARE

## OBJETIVOS DE APRENDIZAGEM

**A partir do estudo desta unidade, você deverá ser capaz de**:

- identificar como podemos fazer para gerar a qualidade na verificação de cada requisito, já fazendo uma análise criteriosa de todas as possibilidades que podemos ter na sua elaboração e execução;
- saber organizar e gerenciar todos os requisitos, buscando sempre manter suas funcionalidades e objetivos ativos;
- criar um ciclo de vida dos requisitos, desde a sua criação até a sua implementação;
- compreender as mudanças feitas nos requisitos no ciclo de desenvolvimento de software, devido às evoluções que acontecem com o tempo;
- entender o mercado, em constante expansão, e o que ele procura nos profissionais dessa área, assim como as ferramentas utilizadas para manipular os requisitos.

## PLANO DE ESTUDOS

A cada tópico desta unidade você encontrará autoatividades com o objetivo de reforçar o conteúdo apresentado.

TEMA DE APRENDIZAGEM 1 – QUALIDADE DE REQUISITOS

TEMA DE APRENDIZAGEM 2 – GERENCIAMENTO DE REQUISITOS

TEMA DE APRENDIZAGEM 3 – O MERCADO DE TRABALHO DE ENGENHARIA DE REQUISITOS

## CHAMADA

Preparado para ampliar seus conhecimentos? Respire e vamos em frente! Procure um ambiente que facilite a concentração, assim absorverá melhor as informações.

# CONFIRA A TRILHA DA UNIDADE 3!

Acesse o
QR Code abaixo:

# QUALIDADE DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, neste tema de aprendizagem, referente à qualidade de requisitos, veremos que a noção clássica de requisitos de qualidade foca na Adequação, sem Ambiguidade, Completude, Consistência, Verificabilidade, Modificabilidade e Rastreabilidade (IEEE 1993). Na prática, porém, a maioria dos requisitos as especificações não atendem a essas qualidades. Alguém poderia argumentam que este é apenas um problema de aplicação do direito métodos e processos e que devemos melhorar nossos processos de requisitos até que eles produzam as qualidades desejadas. No entanto, um olhar mais atento revela que não é tão simples. As próprias qualidades fazem parte o problema.

A noção de completude leva a modelos de processo, em que uma especificação de requisitos do sistema completo deve ser produzida antes de qualquer projeto e atividades de implementação. No entanto, os clientes nem sempre conhecem e entender o que eles querem. Sistemas e requisitos evoluem. Portanto, é quase impossível produzir e congelar uma especificação de requisitos completa. A falta de ambiguidade exige que a especificação seja o mais formal possível. No entanto, na grande maioria das especificações de requisitos, os requisitos são declarados informalmente com linguagem natural ou, na melhor das hipóteses, semiformal, por exemplo, com modelos de classe ou fluxo de dados de modelos. Assim, a inequívoco é muito difícil de alcançar. O valor das faixas de rastreabilidade, na prática, vai de irrelevante (muitos projetos internos) a extremamente importantes (projetos críticos de segurança). Por outro lado, também temos processos e problemas relacionados ao método. Em muitos projetos, os clientes não conseguem avaliar a adequação dos requisitos porque a maneira como os requisitos são representados não corresponde à maneira como os clientes usam um sistema e pense sobre isso. Além disso, quando os clientes não sabem e entendem totalmente o que querem, a avaliação da adequação torna-se ainda mais difícil.

Consequentemente, precisamos tanto de uma mudança no paradigma básico de requisitos de qualidade quanto de uma adaptação adequada de técnicas de engenharia de requisitos, a fim de atender o conjunto modificado de qualidades.

# 2 A IMPORTÂNCIA DE AVALIAR A QUALIDADE DE REQUISITOS

A qualidade de requisitos é o ponto inicial para a qualidade de software. Atualmente, temos várias técnicas onde podemos fazer a avalição da qualidade de requisitos. As técnicas que hoje encontramos disponíveis servem exatamente para prestar e fazer a medição de atributos de qualidade baseados em requisitos tanto funcionais quanto também não-funcionais, inclusive aqueles que tem uma aplicação direta em ambientes com realidade virtual.

Seguindo a indicação de Fabbrini *et al*. (2001), todo outro processo de avaliação, quando nos referimos à avaliação de qualidade dos requisitos de um sistema, aplicado em uma linguagem natural, também irá necessitar de um modelo pré-determinado de qualidade, no qual deverá conter um conjunto de atributos de qualidade.

As partes interessadas descreverão os requisitos de qualidade a qualquer momento, mas é particularmente importante focar neles durante seus esforços iniciais de definição de escopo durante a Iniciação. Segundo Fabbrini *et al*. (2001), considerar os requisitos de qualidade no início do ciclo de vida são importantes porque:

1- Os requisitos de qualidade orientam importantes decisões de arquitetura. Ao identificar sua estratégia de arquitetura, muitas vezes, você descobrirá que são os NFRs que serão os principais impulsionadores de sua arquitetura.
2- Os requisitos de qualidade orientarão alguns aspectos de sua estratégia de teste. Como eles tendem a ser transversais e a direcionar aspectos importantes de sua arquitetura, eles tendem a direcionar aspectos importantes de sua estratégia de teste. Por exemplo, os requisitos de segurança conduzirão a necessidade de oferecer suporte a testes de segurança, os requisitos de desempenho conduzirão à necessidade de testes de estresse e carga e assim por diante. Essas necessidades de teste, por sua vez, podem direcionar aspectos de seus ambientes de teste e suas escolhas de ferramentas de teste.
3- Os requisitos de qualidade direcionarão os critérios de aceitação para os requisitos funcionais (como histórias). Os requisitos de qualidade geralmente abrangem todo o sistema, portanto, eles se aplicam a muitos e, às vezes, a todos os seus requisitos funcionais. Parte de garantir que sua solução seja potencialmente consumível a cada iteração é garantir que ela cumpra suas metas gerais de qualidade, incluindo os requisitos de qualidade aplicáveis. Isso é particularmente verdadeiro com soluções de vida crítica e de missão crítica.

Segundo Rocha *et al*. (2006), qualidade é definida como adequação ao uso e adesão aos requisitos. Ambas as condições devem ser atendidas para alcançar a qualidade. A adequação ao uso é determinada, em última análise, pelo cliente comercial. O patrocinador do projeto é responsável por determinar o que satisfará o negócio e

comissionar o projeto adequado para criá-lo. No entanto, mesmo que um projeto seja concluído no prazo, dentro do orçamento e produza exatamente o que foi solicitado (ou seja, cumpra os requisitos), ainda será considerado um fracasso se o cliente comercial não perceber valor no que o projeto produziu (ou seja, adequação para uso).

A adesão aos requisitos é de responsabilidade do gerente de projeto. O ato de provar se os requisitos foram ou não atendidos é chamado de validação; e normalmente é realizado por meio de observação e medição. Quando os requisitos são definidos corretamente, a validação pode ser alcançada independentemente de quem executa as atividades de validação ou quantas vezes elas são repetidas. Por outro lado, quando os requisitos não são bem definidos, resultados de teste inconsistentes, argumentos, expectativas desalinhadas, partes interessadas insatisfeitas e percepções de falha são mais prováveis.

Sabemos que a qualidade de requisitos é de grande importância dentro do processo de viabilização do software e validação dos requisitos. Temos que entender, também, quais são os requisitos da qualidade de software, como colocamos a seguir.

**ATENÇÃO**

Qualidade de requisitos: esta sistematização só será possível se forem priorizados e atendidos pelo menos quatro requisitos da qualidade de software: usabilidade, confiabilidade, funcionalidade e manutenibilidade (ROCHA *et al*., 2006).

## 2.1 CARACTERÍSTICAS DA DECLARAÇÃO DE REQUISITOS DE QUALIDADE

Como podemos distinguir bons requisitos de software daqueles que apresentam problemas? Na sequência, vamos descrever seis características que as declarações de requisitos de qualidade devem exibir.

Uma qualidade poderosa é escrever casos de teste de acordo com os requisitos antes de cortar uma única linha de código. Casos de teste cristalizam sua visão do comportamento do produto conforme especificado nos requisitos e podem revelar imprecisões, omissões e ambiguidades.

Agora, vamos explicar as sete características da qualidade dos documentos de requisitos (PRESSMAN, 2006):

1- **Correto**: cada requisito deve descrever com precisão a funcionalidade a ser entregue. A referência para correção é a fonte do requisito, como um cliente real ou uma especificação de requisitos de sistema de nível superior. Um requisito de software que entra em conflito com um o requisito do sistema correspondente não está correto (obviamente, a própria especificação do sistema pode estar incorreta). Somente representantes do usuário podem determinar a exatidão dos requisitos do usuário, o que é essencial incluí-los, ou seus substitutos próximos, nas inspeções dos requisitos.

   As inspeções de requisitos que não envolvem usuários podem levar os desenvolvedores a dizer: "Isso não faz sentido. Isso é provavelmente o que eles queriam dizer. Isso também é conhecido como "adivinhar".
2- **Viável**: deve ser possível implementar cada requisito dentro dos recursos conhecidos e limitações do sistema e seu ambiente. Para evitar requisitos inviáveis, tenha um trabalho do desenvolvedor com os analistas de requisitos ou pessoal de marketing durante todo o processo de elicitação.
3- **Processo**: pode fornecer uma verificação da realidade sobre o que pode e o que não pode ser feito tecnicamente, e o que só pode ser feito a um custo excessivo ou com outras compensações.
4- **Necessário**: cada requisito deve documentar algo que os clientes realmente precisam ou algo que é necessário para conformidade com um requisito externo, uma interface externa ou um padrão. Outra maneira de pensar em "necessário" é que cada requisito se originou de uma fonte que você reconhece como tendo autoridade para especificar requisitos. Rastreie cada requisito de volta a sua origem, como um caso de uso, requisito do sistema, regulamentação ou algum outro recurso de voz da entrada do cliente. Se você não consegue identificar a origem, talvez o requisito seja um exemplo de "*gold plating*" e não é realmente necessário.
5- **Priorizado**: atribua uma prioridade de implementação para cada requisito, recurso ou caso de uso para indicar o quão essencial é incluí-lo em uma versão específica do produto. Clientes ou seus os substitutos têm a maior parte da responsabilidade de estabelecer prioridades. Se todos os requisitos são considerados igualmente importantes, o gerente de projeto é menos capaz de reagir a novos requisitos adicionados durante o desenvolvimento, cortes orçamentários, atrasos no cronograma ou a saída de um membro da equipe. A prioridade é uma função do valor fornecido ao cliente, o custo relativo de implementação e o risco técnico relativo associado à implementação.

   Utilizamos três níveis de prioridade. Alta prioridade significa que o requisito deve ser incorporado no próximo lançamento do produto. Prioridade média significa que o requisito é necessário, mas pode ser adiado para uma versão posterior, se necessário. Baixa prioridade significa que seria bom ter, mas caso nós percebermos que pode ter que ser descartado se não tivermos tempo ou recursos insuficientes.
6- **Inequívoco:** o leitor de uma declaração de requisito deve ser capaz de desenhar apenas uma interpretação dela. Além disso, vários leitores de um requisito devem chegar a mesma interpretação. A linguagem natural é altamente propensa à ambiguidade. Evite palavras subjetivas como fácil de usar, fácil, simples, rápido, eficiente, vários, avançado, aprimorado, maximizar e minimizar. Escreva cada requisito em linguagem

sucinta, simples e direta do domínio do usuário, não em computador. Maneiras eficazes para revelar a ambiguidade incluem inspeções formais das especificações de requisitos, teste de redação, casos de requisitos e criando cenários de usuário que ilustram o comportamento esperado de uma porção específica do produto.

7- **Verificável**: veja se você pode criar testes ou usar outras abordagens de verificação, como inspeção ou demonstração, para determinar se cada requisito está devidamente implementado no produto. Se um requisito não for verificável, determinar se foi implementado corretamente é uma questão de opinião. Requisitos que não são consistentes, viáveis ou inequívocos também não são verificáveis. Qualquer requisito que diga que o produto deve "suportar" algo não é verificável.

Essas características são muito importantes para toda a verificação de uma boa qualidade do processo de desenvolvimento dos requisitos. Temos que verificar exatamente a questão das condições de utilização deles.

**ATENÇÃO**

Os requisitos têm que ser definidos a partir das condições de utilização dos produtos, entendendo também os seus objetivos, suas funções ou mesmo o desempenho que é esperado. Então, podemos entender que são fatores relativos e integrados à qualidade de todo o processo de desenvolvimento do produto/software, que são percebidos apenas pelas pessoas nas quais já trabalharam no seu desenvolvimento.

## 2.2 CENÁRIOS DE REQUISITOS DE QUALIDADE

Quando usamos alguns cenários para poder especificar alguns requisitos, isso com certeza tem um grande impacto positivo nas qualidades de adequação do requisito, na completude parcial, na modificabilidade e na verificabilidade – desde que os cenários sejam usados adequadamente.

Os cenários situam os requisitos no ambiente onde um sistema será usado e descrevê-los em uma maneira orientada ao usuário. Junto à decomponibilidade em funções ou transações do usuário e a facilidade de compreensão, temos os ingredientes que permitem uma validação contínua de requisitos escritos contra as intenções dos clientes, resultando assim em especificações adequadas.

Cada cenário (ou grupo de cenários relacionados) representa uma especificação parcial que é coerente de uma perspectiva do usuário. Assim, a completude parcial vem naturalmente com o suporte de especificações parciais.

O uso de cenários leva a um particionamento orientado ao usuário de funcionalidade, tornando mais fácil lidar com a evolução dos requisitos, melhorando assim a modificabilidade.

Além disso, os casos de teste podem ser derivados de cenários em forma natural e direta (RYSER; GLINZ,1999). Assim, os cenários também são verificáveis.

O uso de cenários possibilita processos que usam ciclos curtos entre escrever e validar requisitos e que definem casos de teste derivados dos cenários antecipadamente. Tais processos, juntamente com a orientação do usuário de cenários, produzem recursos poderosos para detectar e resolver ambiguidades. Assim, os cenários não conduzem a especificações a priori não ambíguos (isso ocorre porque eles usam linguagem natural), mas eles suportam processos com proximidade, loops de feedback que são os meios naturais de detectar e resolver ambiguidades na comunicação entre humanos.

A consistência não é promovida pelo uso de cenários. Pelo contrário, ver cada cenário como uma entidade pode levar a graves problemas de inconsistência. Além disso, nem todos os requisitos devem ser descritos por cenários. Por exemplo, os dados persistentes e comportamento dependente do estado são mais fáceis de especificar usando modelos de objetos e autômatos de estado.

Assim, em abordagens baseadas em cenários, o esforço explícito se faz necessário para garantir a consistência. A consistência entre cenários e modelos de objetos pode ser melhorado por meio de referências cruzadas sistemáticas (GLINZ, 2000).

## 2.2.1 Representação de Cenários

Combinando cenários com uma visão estática de um sistema, podemos dizer que nem todo requisito deve ser descrito por cenários. Por exemplo, dados persistentes e comportamento dependente do estado são mais fáceis de especificar usando modelos de objetos e autômatos de estado.

Assim que empregarmos mais de uma técnica de modelagem para descrever os requisitos de um sistema, temos o problema de combinar esses modelos de forma sistemática e consistente. Uma abordagem leve para a consistência pode ser baseada em referência cruzada entre um modelo de cenário e um modelo de objeto. Uma abordagem mais formal requer um modelo que integra cenários, modelos de objetos e modelos de comportamento em um único modelo integrado (GLINZ, 2000).

Segundo Glinz (2000), a composição de cenários baseada em gráficos de estado nos fornece recursos de validação e verificação que vão muito além daqueles disponíveis para um conjunto de cenários. Explorando as propriedades dos diagramas de estado, nós podemos:

- avaliar a adequação de um cenário não apenas isoladamente, mas também em seu contexto;
- verificar se a especificação expressa os requisitos propriedades de um sistema adequadamente, por exemplo, exclusão mútua de cenários que não devem ser executados em paralelo ou a acessibilidade de um determinado estado de interação;
- gerenciar especificações parciais. Também ajuda a detectar cenários ausentes, sobrepostos ou contraditórios.

## 2.3 DIRETRIZES PARA ESCREVER REQUISITOS DE QUALIDADE

Não existe uma fórmula para escrever requisitos excelentes. É em grande parte uma questão de experiência e aprendizado com os problemas de requisitos que você encontrou no passado. Moreira, Araújo e Brito (2002) indicam algumas diretrizes a serem lembradas ao documentar os requisitos de software. São elas:

- Mantenha frases e parágrafos curtos. Use a voz ativa. Use gramática, ortografia e pontuação. Use termos consistentemente e defina-os em um glossário ou dicionário de dados.
- Para ver se uma declaração de requisito está suficientemente bem definida, leia a perspectiva do desenvolvedor. Adicione mentalmente a frase "me ligue quando terminar" ao final do requisito e veja se isso o deixa nervoso. Em outras palavras, você precisaria de mais esclarecimento do autor, para entender o requisito bem o suficiente para projetar e implementar? Em caso afirmativo, esse requisito deve ser elaborado antes de prosseguir com a construção.
- Os autores de requisitos geralmente lutam para encontrar o nível certo de granularidade. Evite narrativa longa e parágrafos que contêm vários requisitos. Uma diretriz de granularidade útil é escrever requisitos testáveis individualmente. Se você puder pensar em um pequeno número de testes relacionados para verificar a implementação correta de um requisito, provavelmente está escrito no nível certo de detalhe. Se você imagina muitos tipos diferentes de testes, talvez vários requisitos tenham sido agrupados juntos e devem ser separados.
- Fique atento a vários requisitos que foram agregados em uma única declaração. Conjunções como "e" e "ou" em um requisito sugerem que vários requisitos foram combinados. Nunca use "e/ou" em uma declaração de requisito.
- Escreva os requisitos em um nível consistente de detalhes em todo o documento. É muito comum vermos algumas especificações de requisitos que variam amplamente em seu escopo. Por exemplo, "um código de cor válido deve ser R para vermelho" e "um código de cor válido deve ser G para verde" pode ser dividido em separado requisitos, enquanto "o produto deve responder às diretivas de edição inseridas por voz" descreve um subsistema inteiro, não um único requisito funcional.
- Evite declarar os requisitos de forma redundante. Embora incluindo o mesmo requisito em vários lugares pode facilitar a leitura do documento, também facilita a manutenção do documento mais difícil. Todas as instâncias múltiplas do requisito devem ser atualizadas em ao mesmo tempo, para que não surja uma inconsistência.

Se você observar essas diretrizes e revisar os requisitos formal e informalmente, cedo e frequentemente, seus requisitos fornecerão uma base melhor para a construção do produto, teste do sistema e satisfação final do cliente. Lembre-se que, sem alta qualidade de requisitos, o software é como uma caixa de chocolates: você nunca sabe o que vai conseguir.

# 3 INSPEÇÕES

A inspeção de requisitos, conforme Pfleeger (2004), é uma técnica desenvolvida por Fagan, em 1972, enquanto desenvolvia seu trabalho na IBM, e a criação visava aumentar a qualidade de software e aumentar a produtividade dos programadores envolvidos no processo. Esta técnica inicialmente teve uma centralização na localização de defeitos na estrutura e também no código de programas. Posteriormente, teve sua ampliação para a aplicação em outros artefatos de software (como requisitos, especificações, arquitetura, planos de teste), e, atualmente, tem bastante utilização por desenvolvedores. Podemos apontar que o processo de inspeção pode fazer a detecção de 30% a 90% dos erros existentes nos artefatos que são gerados num processo de desenvolvimento de software. O processo de inspeção, então, se caracteriza devido à utilização de uma técnica de leitura aplicável a um artefato. Assim, iremos buscar a localização de erros ou defeitos no mesmo, segundo um critério pré-estabelecido. Essa inspeção será aplicada a praticamente todos os tipos de artefatos, e também é possível a utilização de diferentes técnicas de leitura.

## 3.1 MELHORES PRÁTICAS DA INSPEÇÃO DE QUALIDADE DE REQUISITOS

Pfleeger (2004) ainda afirma que um processo de inspeção abrangente envolve as seguintes práticas recomendadas significativas, como:

Documentação dos padrões de qualidade do produto: a documentação é uma parte essencial dos processos de gerenciamento de qualidade, pois ajuda suas equipes de qualidade, auditores e fornecedores durante verificações, auditorias e inspeções de qualidade. A documentação de todas as tarefas de gerenciamento de qualidade também demonstra o compromisso de sua organização com as melhores práticas e cultura de qualidade.

Desenvolva fluxos de trabalho das inspeções: fluxos de trabalho de inspeção configuráveis devem ser desenvolvidos para que sua equipe de inspeção precise de treinamento mínimo. Fluxos de trabalho configuráveis ajudam significativamente a melhorar a produtividade de sua equipe e diminuir o tempo de ciclo do processo de inspeção.

Definir métodos para problemas de qualidade de produtos: Cada *recall* de produto devido a problemas de qualidade do produto não requer um grande esforço. Alguns problemas de qualidade do produto podem ser facilmente corrigidos com o retrabalho. Assim, você pode eliminar o desperdício de recursos determinando os procedimentos de contenção para tais produtos e categorizando os problemas de qualidade para facilitar a identificação.

Documente os incidentes e seu plano de ação: Documente os incidentes e o plano de ação implementado e notifique a gerência para tomar decisões de negócios informadas. Dessa forma, você pode agilizar e transformar os processos de inspeção existentes.

Padronize as inspeções e cumpra os requisitos de conformidade: a padronização das inspeções para cada estágio da produção (pré, pós e produção em processo), ao mesmo tempo em que cumpre os padrões regulatórios aplicáveis, garante problemas mínimos de conformidade. Também promove uma cultura de qualidade e melhorias contínuas dentro da organização, permitindo que você alcance os objetivos de negócios com eficiência.

O momento-chave do ponto de vista do autor se tata dos atributos de qualidade concretos e medidas concretas. Sem um acordo sobre quais atributos são essenciais e como medir eles, toda qualidade será apenas uma caixa vazia, sem a possibilidade de utilização na situação concreta. O problema de atributos e medidas é o problema geral problema da qualidade do produto de requisitos de software. A possível realização do projeto depende fortemente da capacidade da comunidade de padronização de software para compartilhar a experiência nas medidas de qualidade de requisitos do software usadas pelas softwarehouses de vanguarda e qualidade de software especialistas e sobre o acordo sobre quais medidas serão ser selecionado para a formulação de requisitos de qualidade, a medição consecutiva da qualidade do software e avaliação e classificação de qualidade.

A esta altura, a criticidade da gestão da qualidade para o sucesso dos projetos deve ser evidente. Sem requisitos bem definidos e com linha de base, o sucesso do projeto permanece vulnerável a opiniões subjetivas. O próprio ato de criar requisitos de qualidade revela onde diferentes entendimentos podem existir no início do ciclo de vida do projeto. Isso reduz muito o risco de o cliente esperar algo diferente do que foi entregue, depois de gastos todos os recursos orçados. Por todas essas razões (e mais), um plano de qualidade de requerimento do projeto é uma parte essencial de qualquer plano de projeto profissional.

Como nós falamos de qualidade de requisitos, é muito importante também citar algumas normas para fazer toda a parte de verificação. O teste baseado em requisitos e seu processo inerente de rastreabilidade e verificação de requisitos são amplamente vistos como uma prática recomendada de acordo com os padrões corporativos, como

o Capability Maturity Model Integration (CMMI). A CMMI é uma descrição do nível de esclarecimento de um processo. É essencialmente uma medida da qualidade e capacidade de um processo. Existem cinco categorias, em uma das quais cada processo se enquadrará.

Identificamos que existem cinco níveis do CMMI. Tecnicamente, existem seis níveis, quando você inclui o nível zero. O nível 0 é "indefinido" pelo CMMI e representa um processo *ad hoc*, ou falta de processo.

Níveis CMMI
CMMI Nível 0. Indefinido. Nenhum processo real.
CMMI Nível 1. Realizado. Um processo é definido, mas desorganizado.
CMMI Nível 2. Gerenciado. Um processo definido é gerenciado.
CMMI Nível 3. Definido. Um processo gerenciado é padronizado em toda a empresa.
CMMI Nível 4. Medida quantitativamente. O desempenho de um processo padronizado é medido.
CMMI Nível 5. Otimização. A medição de desempenho é usada como um loop de feedback para melhorar o processo.

Uma outra norma de melhoria de processos de software e de requisitos é o MPS.br (Melhoria do Processo de Software Brasileiro), que se trata de uma metodologia voltada essencialmente à área de desenvolvimento de sistemas. Foi criada através de um conjunto de organizações ligadas ao desenvolvimento de software. Falando nas instituições que criaram o MPS.br, podemos citar: a Softex (SP), a RioSoft (RJ), a empresa COPPE/UFRJ (RJ) e a CESAR (PE). Na verdade, todas essas organizações quase na sua totalidade são normalmente não-governamentais e muitas das vezes tem sua origem na área acadêmica, possuindo também um destaque grande junto a toda comunidade de software brasileira.

Temos que destacar que, dentro do MPS-BR, faz-se o uso das principais e mais importantes abordagens internacionais voltadas para a sua definição, a sua avaliação e para a melhoria dos processos de software. Devido esses fatos, o MPS-BR torna-se muito mais compatível, inclusive com as práticas do CMMI. Há ainda no MPS-BR uma certa estrutura em relação aos níveis de maturidade, de uma forma muito similar àquela que já é executada dentro do CMMI.

# RESUMO DO TÓPICO 1

**Neste tópico, você aprendeu:**

- A importância de avaliar um requisito de qualidade, determinando se temos um problema de requisitos e saber também o tamanho desse problema para calcular com precisão a quantidade de esforço necessária para transformar nossos recursos insuficientes em recursos satisfatórios.

- Todas as características que um requisito de qualidade precisa apresentar para facilitar o desenvolvimento do software.

- Como que os cenários para a especificação de requisitos têm um forte impacto positivo nas qualidades de adequação, completude parcial, modificabilidade e verificabilidade – desde que os cenários sejam usados adequadamente.

- Como a inspeção se caracteriza-se, fazendo a utilização de uma técnica de leitura aplicável a um artefato, buscando a localização de erros ou defeitos no mesmo, segundo um critério pré-estabelecido.

# AUTOATIVIDADE

1 Quando falamos da importância de avaliar um requisito de qualidade, levando em consideração os diversos cenários, estabelecemos um benchmark para nossas especificações de requisitos quanto ao valor de uma métrica de critérios de qualidade. Definimos quatro valores para refletir cada situação. Sobre esses valores, analise as sentenças a seguir:

I- Elaborar um processo original em um ambiente difícil não é tarefa fácil.
II- Elaborar uma narrativa inovadora sem qualquer pressão ou limitação.
III- Desenvolvimento mundano.
IV- Elaborar um processo de narrativa formal através de requisitos de precisão.

Assinale a alternativa CORRETA:
a) ( ) Somente a sentença I está correta.
b) ( ) As sentenças III e IV estão corretas.
c) ( ) As sentenças I, II e IV estão corretas.
d) ( ) As sentenças I, II e III estão corretas.

2 Como podemos distinguir bons requisitos de software daqueles que apresentam problemas? Uma qualidade poderosa é escrever casos de teste de acordo com os requisitos antes de cortar uma única linha de código. Casos de teste cristalizam sua visão do comportamento do produto conforme especificado nos requisitos e podem revelar imprecisões, omissões e ambiguidades. Existem seis características da qualidade dos documentos de requisitos, uma delas é termos requisitos viável. Sobre requisitos viáveis, assinale a alternativa CORRETA:

a) ( ) Cada requisito deve descrever com precisão a funcionalidade a ser entregue.
b) ( ) Deve ser possível implementar cada requisito dentro dos recursos conhecidos e limitações do sistema e seu ambiente.
c) ( ) Este desenvolvedor pode fornecer uma verificação da realidade sobre o que pode e o que não pode ser feito tecnicamente, e o que só pode ser feito a um custo excessivo ou com outras compensações.
d) ( ) Cada requisito deve documentar algo que os clientes realmente precisam ou algo que é necessário para conformidade com um requisito externo, uma interface externa ou um padrão.

3 O leitor de uma declaração de requisito deve ser capaz de desenhar apenas uma interpretação dela. Além disso, vários leitores de um requisito devem chegar a mesma interpretação. A linguagem natural é altamente propensa à ambiguidade. Evite palavras subjetivas como fácil de usar, fácil, simples, rápido, eficiente, vários, avançado, aprimorado, maximizar e minimizar. Escreva cada requisito em linguagem sucinta, simples e direta do domínio do usuário, não em computador. Sobre qual característica de declaração de requisitos de qualidade estamos nos referindo?

a) ( ) Correto.
b) ( ) Viável.
c) ( ) Necessário.
d) ( ) Inequívoco.

4 O processo de inspeção, então, se caracteriza devido à utilização de uma técnica de leitura aplicável a um artefato. Explique pelo menos duas boas práticas recomendadas que são significativas de um processo de inspeção.

5 A composição de cenários baseada em gráficos de estado nos fornece recursos de validação e verificação que vão muito além daqueles disponíveis para um conjunto de cenários. Explorando as propriedades dos diagramas de estado, nós podemos ter algumas funcionalidades. Neste contexto, disserte sobre quais funcionalidades que podemos ter em relação à validação e verificação dos gráficos de estado.

# GERENCIAMENTO DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 2, iremos ver como faremos para gerenciar os requisitos no desenvolvimento do software. O gerenciamento de requisitos é o processo de reunir, analisar, verificar e validar as necessidades e requisitos para um determinado produto ou sistema que atualmente está em desenvolvimento. Fazer um bom gerenciamento irá garantir que as entregas finalizadas irão atender às expectativas das partes envolvidas. Podemos gerenciar os requisitos usando documentos, no entanto, sistemas ou produtos complexos em setores altamente regulamentados reduzem o risco usando ferramentas confiáveis de gerenciamento de requisitos. O gerenciamento de requisitos normalmente também inclui gerenciamento de mudanças para requisitos, rastreabilidade de requisitos, colaboração com partes interessadas relevantes e aprovações de requisitos.

Segundo Coventry (2015), o Gerenciamento de Requisitos é um conjunto iterativo de atividades que ajudam a garantir que a elicitação, da documentação, o refinamento e as mudanças de requisitos sejam tratadas adequadamente durante o ciclo de vida, com o objetivo de satisfazer a missão ou necessidade geral de maneira qualitativa e para a satisfação do cliente.

Essa definição de Dorfman e Thayer (1997) em relação à engenharia de requisitos de software inclui elicitação, análise, especificação, verificação e gerenciamento do software de requisitos, com o último sendo o planejamento e controle de todas essas atividades relacionadas. Todas essas atividades estão incorporadas na definição de gerenciamento de requisitos. A diferença está principalmente na escolha da palavra "gerenciamento" em vez de "engenharia". Gestão é uma descrição mais apropriada de todas as atividades envolvidas e enfatiza com precisão a importância de acompanhar as mudanças para manter acordos entre aquelas partes que desejam o projeto e a equipe do projeto. Muitos não estão familiarizados com o termo “elicitação”, que tem uma definição de um conjunto de atividades que as equipes irão empregar para obter ou descobrir solicitações das partes interessadas, determinar as necessidades reais por trás das solicitações e chegar a um conjunto adequado de requisitos que podem ser colocados no sistema para atender a essas necessidades.

Isso é feito continuamente durante todo o ciclo de vida do produto . O gerenciamento de requisitos tem suas raízes na engenharia de sistemas, mas também pode ser aplicado em diversas disciplinas, como análise de negócios e gerenciamento de projetos. Vamos considerar o gerenciamento de requisitos para gerentes de produto e equipes de desenvolvimento de software. Normalmente, um gerente de produto é o proprietário do processo de gerenciamento de requisitos. Você é responsável por ajudar a equipe a definir os requisitos do produto e gerenciar as mudanças ao longo do desenvolvimento. Quando você aprende como gerenciar requisitos com eficiência, pode validar melhor as necessidades do cliente e criar recursos que os clientes consideram adoráveis.

## 2 POR QUE GERENCIAR REQUISITOS?

O gerenciamento de requisitos é importante porque capacita todos a entender claramente as expectativas das partes interessadas e entregar com confiança um produto que foi verificado para atender aos requisitos e validado para atender às necessidades.

Segundo Machado (2011), a gerência de requisitos vai incluir uma documentação de dependências através dos requisitos, também um controle de mudanças entre as devidas identificações e suas correções de inconsistências entre duas situações, o próprio requisito e alguns artefatos de um projeto em questão.

Já Pressman (2006) afirma que é fundamental o gerenciamento de alterações daqueles requisitos acordados, fazer o gerenciamento de 2 relacionamentos entre requisitos e o gerenciador de dependências entre requisitos.

Wiegers (2003) discorre que o gerenciamento de requisitos inclui algumas atividades, tais como: o controle de mudanças, um controle de versão, a questão de acompanhamento do estado dos requisitos presentes e também o rastreamento de requisitos.

Podemos entender melhor a necessidade de gerenciar os requisitos, confira a figura a seguir:

Figura 1 – Requisitos de Sistema


Fonte: https://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx. Acesso em: 19 abr. 2023.

O gerenciamento de requisitos é um processo sofisticado que inclui muitas partes móveis e diversos grupos de pessoas. Normalmente, o departamento de gerenciamento de produtos, especificamente o gerente de produtos, é responsável pelo processo de gerenciamento de requisitos. Eles trabalham com as partes interessadas, incluindo equipes de negócios, clientes, usuários, desenvolvedores, testadores, reguladores e garantia de qualidade.

Além disso, um produto pode ter apenas 100 requisitos ou vários milhares. Esse número dependerá da complexidade do produto e do nível de regulamentação. Com todos esses elementos em jogo, o sucesso depende da capacidade de colocar todos na mesma página, trabalhando para o mesmo objetivo.

Portanto, o valor comercial do gerenciamento de requisitos é enorme, pois o gerenciamento de requisitos bem-sucedido é a chave para o sucesso do projeto. Segundo Travassos, Gurov e Amaral (2002), os benefícios do gerenciamento de requisitos incluem:

- Melhorar a compreensão das necessidades, requisitos e expectativas das partes interessadas e o problema ou oportunidade que o produto pretende abordar.
- Obtenha clareza sobre escopo, orçamento e cronograma.
- Redução de custo.
- Melhorando a qualidade.
- Diminuindo o tempo gasto.
- Diminui riscos.
- Habilitando o gerenciamento de escopo eficaz.

A importância do gerenciamento de requisitos é intensificada, no entanto, ao criar produtos complexos ou altamente regulamentados. Isso ocorre porque mais tempo e orçamento são investidos no desenvolvimento. O custo de errar – seja dinheiro, tempo ou reputação – é muito alto para arriscar. Assim, desenvolvedores em setores regulamentados, ou aqueles que desenvolvem produtos com uma longa lista de necessidades e requisitos, tendem a contar com ferramentas de gerenciamento de requisitos, para manter seus projetos em ordem.

Embora possa parecer que gerenciamento de requisitos e gerenciamento de projetos são sinônimos, há uma diferença. Simplesmente, o gerenciamento de projetos consiste em construir o produto dentro do orçamento e cronograma com os recursos disponíveis. O gerenciamento de requisitos garante que o produto seja o produto certo e que seja construído corretamente.

Rosenberg *et al*. (1998) dizem que as organizações existem para entregar produtos e/ou serviços, que são entregues por meio de processos de negócios que são decompostos em requisitos. Os requisitos devem ser gerenciados na estratégia organizacional (portfólio, programas e projetos) ou gerenciamento de operações (negócios do dia a dia). Frequentemente, falta gerenciamento de requisitos no planejamento estratégico, portfólios e iniciativas de negócios do dia a dia (muitas vezes referido como melhoria contínua).

Planejamento Estratégico (da estratégia à execução) – requisitos de negócios (resultados e benefícios) são definidos e quantificados

Portfólio – os requisitos de negócios são usados para vincular a estratégia ao portfólio de mudança

Programa – os requisitos de negócios são usados para definir o escopo do programa e os critérios de sucesso

Projeto – os requisitos do negócio e das partes interessadas são usados para definir o escopo do projeto e os critérios de sucesso. Solução – requisitos não funcionais e de transição são usados para construir e implementar uma solução

Melhoria Contínua (negócios do dia a dia) – os requisitos de negócios e os requisitos das partes interessadas são usados para definir a melhoria contínua, o escopo e os critérios de sucesso. Os requisitos não funcionais e de transição da solução são usados para criar e implementar uma solução.

Quando falamos em gestão de requisitos, temos que lembrar que existe um padrão de fases que compõe o modelo de gestão, executando o controle de mudanças, elaborando o controle de versões, acompanhando o estado dos requisitos e rastreando os requisitos, como na figura a seguir:

Figura 2 – Atividades da gerência de requisitos

| Gerencia de Requisitos | | | |
|---|---|---|---|
| **Controle de Mudanças**<br><br>• Propor mudanças;<br>• Analisar impactos;<br>• Tomar decisões;<br>• Atualizar documentos de requisitos;<br>• Atualizar plano de projeto. | **Controle de versão**<br><br>• Definir o esquema de identificação de versão;<br>• Identificar a versão do documento de requisitos;<br>• Identificar a versão de cada requisito. | **Acompanhar o estado de requisitos**<br><br>• Definir possíveis estados para um requisito;<br>• Armazenaro estado de cada requisito;<br>• Documentaros estados de todos os requisitos. | **Rastrear requisitos**<br><br>• Definir ligações com outros requisitos;<br>• Definir ligações com outros elementos. |

Fonte: adaptado de Wiegers (2003)

Quando você planeja o gerenciamento de requisitos? A qualquer momento – nunca é tarde demais para fazer um plano de gerenciamento de requisitos. A situação ideal é gerenciar os requisitos do plano estratégico, que muitas vezes são expressos como resultados e benefícios, até a execução. No entanto, a realidade é que, na maioria das vezes, o gerenciamento de requisitos não é feito até o início de um projeto. Uma das razões pelas quais muitas organizações falham em perceber o valor de uma boa estratégia é a falta de gerenciamento de requisitos com percepção de benefícios. Portanto, fazer o controle de mudanças, indicando uma proposta de mudança, analisando todos os impactos que podem trazer, executando assim uma tomada de decisão, já atualizando os documentos de requisitos e o plano de projeto se torna muito importante nessas situações.

O controle das versões se torna também importante nesse momento, onde teremos que fazer a definição de todo o esquema de identificação das versões correntes, assim como dos documentos de requisitos.

Um outro aspecto interessante é fazer o acompanhamento do estado dos requisitos, fazendo a definição, o armazenamento e a documentação de todos os estados dos requisitos.

Por fim, temos o rastreamento dos requisitos para verificar a ligação que ele terá com os outros requisitos e com outros elementos, onde falaremos um pouco mais até o final desse documento.

Existe algum momento em que você não precisa de um gerenciamento de requisitos? Provavelmente não, mas é importante variar os níveis de detalhamento em um plano de gerenciamento de requisitos, dependendo da complexidade e do risco do projeto.

Dependendo do tipo de requisito, sua organização pode ter vários proprietários e partes interessadas. Normalmente, um gerente de produto é responsável pelo processo de gerenciamento de requisitos no que se refere ao produto e seus recursos.

Uma grande parte de sua função é coletar ideias e feedback de clientes, equipes internas e outras partes interessadas. Você deve avaliar se as novas ideias se alinham com a estratégia geral de negócios e produtos e, em seguida, traduzir as ideias em recursos e requisitos.

Pode parecer óbvio que a engenharia tem um grande papel no gerenciamento de requisitos. Você trabalhará com equipes multifuncionais para refinar os requisitos, e com outras equipes também. Por exemplo: você provavelmente trabalhará com a equipe de finanças ou de operações de negócios para garantir que possa medir e relatar os requisitos de negócios.

Para evitar que um conjunto de requisitos se sobreponha a outro, a comunicação constante entre as partes interessadas é fundamental. Como gerente de produto, você está no centro, supervisionando os requisitos que afetam o negócio, seus clientes, o produto e a equipe de produto.

Então, o que pode ser difícil em um processo destinado a garantir que um sistema esteja em conformidade com as expectativas definidas para ele? Quando colocar na prática em projetos reais, as dificuldades vêm à tona. Segundo Coventry (2015), uma lista mais abrangente de problemas inclui:

- Os requisitos nem sempre são óbvios e têm muitas fontes.
- Os requisitos nem sempre são fáceis de expressar claramente em palavras.
- Existem muitos tipos diferentes de requisitos em diferentes níveis de detalhamento.
- O número de requisitos pode se tornar incontrolável se não for controlado.
- Os requisitos estão relacionados entre si e com outras entregas do processo de várias maneiras.
- Os requisitos têm propriedades únicas ou valores de propriedade. Por exemplo, eles são nem igualmente importantes, nem igualmente fáceis de encontrar.
- Existem muitas partes interessadas e responsáveis, o que significa que os requisitos precisam ser gerenciados por grupos interfuncionais de pessoas.
- Mudança de requisitos.
- Os requisitos podem ser sensíveis ao tempo.

Quando esses problemas são combinados com gerenciamento inadequado de requisitos e habilidades de processo, e a falta de ferramentas fáceis de usar, ferramentas, muitas equipes se desesperam por nunca conseguirem gerenciar bem os requisitos.

## 2.1 CICLO DE VIDA DE REQUISITOS

Segundo Coelho (2014), o ciclo de vida do requisito envolve várias fases e, às vezes, pode ser um processo complicado. A natureza do processo depende da metodologia que você escolher para o desenvolvimento de software, como Agile, Waterfall, Incremental etc. Cada fase pode envolver muita papelada e procedimentos de aprovação. Ele também lida com os documentos do projeto, como uma proposta de projeto, plano de gerenciamento do projeto, escopo do projeto e caso de negócios.

Já para Sammerville (2007), o ciclo de vida dos requisitos é uma abordagem de forma sistematizada para fazer o gerenciamento e manter os requisitos do seu início até a sua implementação final. Neste processo inclui o monitoramento, o seu planejamento, sua análise, o modo de gerenciamento e a comunicação dos seus requisitos organizacionais. Os requisitos se tornam essenciais para qualquer empresa, com o objetivo de ter uma tomada de decisão e assim ter uma permanência mais competitiva. Quando utilizamos o processo de gerenciamento do ciclo de vida dos requisitos (LCM), garantimos que seus requisitos sejam gerenciados e mantidos adequadamente durante todo o ciclo de vida.

**INTERESSANTE**

Os requisitos, dentro do conceito de ciclo de vida, tanto de produtos, processos e sistemas nos permitem (através de seu entendimento) a melhor aplicação de testes de verificação e validação.

O objetivo do gerenciamento do ciclo de vida dos requisitos é garantir que os requisitos e designs de negócios, partes interessadas e soluções estejam alinhados entre si e que a solução os implemente.

Segundo Zahran (1998), o ciclo de vida dos requisitos:

- começa com a representação de uma necessidade de negócio como um requisito
- tem sua continuidade através do desenvolvimento de uma possível solução;
- termina quando uma solução e os requisitos que a representam são finalizados.

A gestão de requisitos não termina quando uma solução é implementada. Ao longo da vida de uma solução, os requisitos continuam a agregar valor quando são gerenciados adequadamente. Ainda segundo Zahran (1998), a área de conhecimento do gerenciamento do ciclo de vida dos requisitos faz a inclusão das seguintes tarefas:

- Requisitos de rastreamento.
- Manter requisitos.
- Priorizar requisitos.
- Avaliar Mudanças de Requisitos.
- Aprovar requisitos.

Vamos agora mostrar e explicar cada tarefa do ciclo de vida dos requisitos:

**Requisitos de rastreamento:**

Conforme Rosenberg *et al*. (1998), o propósito de Rastrear Requisitos é garantir que os requisitos e designs em diferentes níveis estejam alinhados entre si e gerenciar os efeitos da mudança em um nível nos requisitos relacionados.

- A rastreabilidade de requisitos identifica e documenta a linhagem de cada requisito, incluindo sua rastreabilidade regressiva, sua rastreabilidade futura.
- Estabelece relação com outros requisitos.
- A rastreabilidade é usada para ajudar a garantir que a solução esteja em conformidade com os requisitos.
- análise de impacto mais rápida e simples.
- descoberta mais que confiável das suas inconsistências e das suas lacunas dentro dos requisitos.
- insights mais profundos sobre o escopo e a complexidade de uma mudança.
- avaliação confiável de quais requisitos foram atendidos e quais não foram. A saída das tarefas são os requisitos (rastreados) e Projetos (rastreados).

**Manter Requisito:**

Segundo Rosenber *et al*. (1998), as razões pelas quais mantemos os requisitos como analista de negócios é porque queremos garantir a precisão deles, além de mantê-los atualizados. O objetivo de manter requisitos é manter a precisão e a consistência deles durante todo o ciclo de vida e oferecer suporte à reutilização em outras soluções.

A manutenção dos requisitos implica garantir a reutilização, mantendo os atributos enquanto se concentra em acompanhar as mudanças nos atributos.

As saídas primárias são requisitos e projetos, que são mantidos e atualizados durante todo o ciclo de vida do projeto.

**Priorizar Requisito:**

Para Rosenberg *et al*. (1998), o propósito de Priorizar Requisitos é classificar os requisitos em ordem de importância relativa. A razão pela qual priorizamos os requisitos se deve ao fato de que as organizações enfrentam restrições e é importante entender os requisitos críticos que levarão à satisfação das partes interessadas.

Classificação com base em seu valor relativo para o negócio.

A seguir está a lista de vários elementos diferentes que devem ser considerados ao priorizar os requisitos:

- Base de Priorização – priorize com base no Benefício, Penalidade, Custo, Complexidade, Sensibilidade de Tempo, dependência, regulamentação ou conformidade.
- Desafios com Priorização – priorização é uma avaliação de valor relativo. Cada parte interessada pode ver o valor de forma diferente. Quando isso ocorre, pode haver conflito entre as partes interessadas. As partes interessadas também podem ter dificuldade em caracterizar qualquer requisito como uma prioridade mais baixa, e isso pode afetar a capacidade de fazer as compensações necessárias. Além disso, as partes interessadas podem (intencionalmente ou não) indicar prioridade para influenciar o resultado de acordo com seus desejos. As compensações do analista de negócios podem ajudar nessas situações.
- Priorização contínua – por exemplo, quando as partes interessadas priorizam no início baseada nos seus benefícios. Com isso, a equipe que faz a implementação tem a opção de fazer uma repriorização dos requisitos baseados em uma sequência que deveram serem implementados devido ao fato de terem restrições técnicas. Assim que a equipe de implementação fornece o devido custo de cada requisito, todas aquelas partes interessadas poderão fazer uma redefinição das suas prioridades novamente.
- Por fim, a saída é requisito e design priorizado.

**Avalie as mudanças nos requisitos:**

O grande propósito de se fazer uma avaliação de mudanças de requisitos é questionar quais implicações daquelas mudanças que foram propostas para os requisitos e os projetos. Ao avaliar as mudanças, os analistas de negócios consideram se cada mudança proposta:

- se alinha com a estratégia geral;
- afeta o valor entregue aos grupos de negócios ou partes interessadas;
- afeta o tempo para entregar ou os recursos necessários para entregar o valor;
- irá alterar a possibilidade de ter quaisquer riscos, oportunidades ou mesmo algumas restrições associadas a uma iniciativa geral (ROSENBERG *et al.*, 1998).

Elementos como formalidade de avaliação, análise de impacto e resolução de impacto serão considerados ao avaliar as mudanças no requisito.

As saídas são Avaliação de Mudança de Requisitos e Avaliação de Mudança de Projeto.

**Aprovar Requisitos:**

Ainda conforme Rosenberg *et al*. (1998), o propósito de Aprovar Requisitos é obter acordo e aprovação de requisitos e projetos para o trabalho de análise de negócios continuar e/ou a construção da solução prosseguir.

Os analistas de negócios têm como responsabilidade fazer a garantia de se ter uma comunicação clara e objetiva de requisitos, dos projetos e de outras informações da sua análise de negócios para que assim suas principais partes interessadas sejam responsáveis pela aprovação dessas informações.

A aprovação de requisitos e projetos pode ser formal ou informal. Abordagens preditivas tem por sua normalidade executar aprovações na sua fase final ou mesmo em meio as reuniões planejadas de seu controle de mudanças.

As abordagens adaptativas normalmente aprovam os requisitos apenas quando a construção e a implementação de uma solução que atende ao requisito podem começar.

Para obter aprovação, como analistas de negócios, devemos entender as funções das partes interessadas, seguir o processo de aprovação, construir consenso e rastrear as aprovações. Às vezes, isso também pode exigir a resolução de conflitos para obter a adesão das partes interessadas.

A saída principal desta tarefa é Design e requisitos aprovados.

## 2.2 RASTREABILIDADE DE REQUISITOS

Os engenheiros e desenvolvedores de software precisam de ferramentas especializadas de gerenciamento de projetos para rastrear os testes e garantir que o produto atenda a todos os requisitos do cliente. Uma dessas ferramentas é o rastreamento de requisitos, que pode ser particularmente útil para projetos grandes e complicados. Você pode usar o rastreamento de requisitos como engenheiro ou desenvolvedor para registrar seu trabalho, coordenar com sua equipe ou mostrar que atendeu aos requisitos legais.

Segundo Kannenberg e Saiedian (2009), a rastreabilidade de requisitos faz referência a sua capacidade de fazer uma descrição e ao seu acompanhamento da vida deste requisito, tanto de um lado frontal quanto de um lado traseiro (ou seja, desde o seu início, passando assim pelo seu desenvolvimento e especificação, até que se faça a sua implantação e seu uso subsequentes, e sabendo de todos os períodos do seu refinamento contínuo e da sua iteração em qualquer uma dessas fases.

Uma situação bem interessante quando falamos de rastreabilidade de requisitos são as ações que ela tem em relação ao seu próprio gerenciamento, como vemos a seguir.

**NOTA**

**Gerenciamento da Rastreabilidade de Requisitos (SOMMERVILLE; SAWYER, 1997):**

Faz a compreensão da origem dos **requisitos**.
Implementa o gerenciamento de todas as mudanças nos **requisitos**.
Indica uma avaliação dos impactos que a mudança de um **requisito** no projeto pode trazer.
Avaliação do impacto que a falha de um teste faz nos **requisitos**.
Verificação de **todos** os **requisitos** do sistema, indicando se foram ou não implementados.

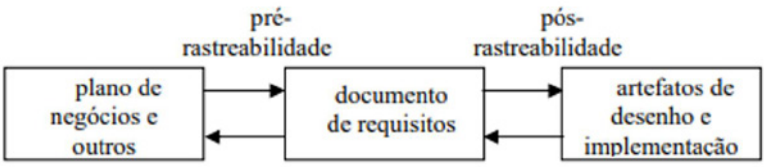
A rastreabilidade de requisitos tem sido identificada na literatura como fator de qualidade, característica que um sistema pode possuir e incluir como um requisito não funcional (RAMESH; JARKE, 2001).

Outa definição diz que rastreabilidade pode ser definida como sendo a técnica usada para prover relacionamento entre requisitos, arquitetura e implementação final do sistema.

Ela irá nos auxiliar na compreensão dos relacionamentos existentes entre requisitos do software ou também entre os artefatos de requisitos, arquitetura e implementação. Esses relacionamentos vão permitir aos projetistas mostrar que realmente o projeto irá atender aos requisitos. A rastreabilidade também tem o papel de apoiar a detecção precoce daqueles requisitos não atendidos pelo software.

A rastreabilidade de requisitos pode inclusive ser vista como uma habilidade de acompanhar e descrever a vida de um requisito, em ambas as direções. A rastreabilidade inicial documenta o contexto a partir do qual emergem os requisitos; e a pós-rastreabilidade vincula os requisitos ao desenho do sistema e sua implementação, conforme a indicação de Davis (1993). A Figura 3 mostra como as ligações possibilitam acompanhar a “vida” de um requisito, em ambas as direções.

Figura 3 – Rastreabilidade de Requisitos




Fonte: Davis (1993, p. 78)

Segundo Davis (1993), a realização de uma análise de rastreabilidade de requisitos é uma parte importante do processo de engenharia de software, pois garante que todos os requisitos foram considerados adequadamente durante cada fase do projeto e que não existem 'buracos' de escopo no sistema desenvolvido devido a requisitos perdidos. A atividade também garante que todos os requisitos sejam internamente consistentes uns com os outros e apoiem os direcionadores, metas e objetivos de negócios abrangentes.

Ainda conforme Davis (1993), a forma mais comum de garantir a total rastreabilidade dos requisitos é por meio de uma Matriz de Rastreabilidade de Requisitos (RTM). A matriz de rastreabilidade normalmente é utilizada para fazer a verificação se todos os requisitos declarados e seus derivados estão associados aos seus respectivos elementos de design correspondentes, também aos componentes do sistema, os módulos e as outras entregas do projeto. Isso é conceituado como sendo um rastreamento direto. O RTM pode também ser usado para fazer a verificação e a documentação da fonte original dos requisitos para que, se surgirem dúvidas do cliente sobre o motivo da inclusão de determinados recursos, haja uma trilha de auditoria abrangente. Isso é conhecido como rastreamento reverso.

A própria matriz de rastreabilidade de requisitos pode ser criada usando ferramentas simples, como uma planilha ou até mesmo em papel. No entanto, para evitar a necessidade de manter vários artefatos separados, uma ferramenta integrada de gerenciamento de requisitos geralmente é a melhor escolha.

Você também pode escolher um RTM especializado para uma determinada função. Uma matriz de gerenciamento de riscos pode ajudá-lo a demonstrar as ferramentas específicas que você está usando para mitigar um risco estabelecido. Uma matriz de conformidade pode ser um RTM que você envia para atender aos requisitos legais se estiver criando ferramentas para um setor em que a segurança ou a qualidade são muito importantes, como transporte ou área médica (DAVIS, 1993).

Tais sistemas permitem gerenciar a lista de requisitos e associar os demais artefatos do projeto (casos de teste, casos de uso, tarefas, defeitos, revisões de código-fonte, apresentações, notas de entrevista etc.) a eles em um ambiente colaborativo. Quando você pode vincular todos esses itens em ambas as direções, você tem o que é conhecido como rastreabilidade de ponta a ponta.

Dorfman e Tahyer (1997) apontam alguns dos benefícios que os requisitos de rastreamento podem fornecer:

1- **Eficiência:** Com um procedimento de rastreamento de requisitos, você pode economizar tempo rastreando os requisitos no mesmo local para poder consultar essas informações sempre que precisar. O uso de uma matriz pode tornar essas informações muito visuais para que você possa encontrá-las rapidamente. Você também pode evitar testes duplicados rastreando os resultados no mesmo documento, para que todos os testadores saibam o que deve ser feito.
2- **Teste de componente isolado:** Um RTM mostra quais componentes de código e design correspondem a requisitos específicos, para que os desenvolvedores possam testar apenas esses requisitos ou seções para entender se eles funcionam. Isso pode ajudar a detectar e corrigir bugs em um estágio anterior do desenvolvimento e localizar um problema na codificação.
3- **Análise de impacto mais fácil:** Se você encontrar desafios ou erros, poderá ver mais facilmente a quais outras partes do projeto eles estão conectados. À medida que você corrige certas seções de código, pode ver rapidamente no RTM se essas seções de código influenciam outros aspectos do design.
4- **Documentação de conformidade regulamentar:** Se o seu projeto estiver sujeito a regulamentos ou políticas da empresa, rastrear os requisitos pode ajudá-lo a cumprir todos os regulamentos. Se você tiver uma lista dos requisitos para cada regulamento, lembre-se de incorporá-los todos. Em algumas indústrias, existem requisitos legais para registrar a rastreabilidade dos requisitos durante a produção e relatar os resultados para garantir que produtos como carros, equipamentos industriais, aviões e outras máquinas poderosas tenham software seguro e bem testado.
5- **Melhor comunicação da equipe:** Ter um RTM comumente disponível pode ajudar sua equipe a colaborar de forma mais eficaz, atualizando todos sobre o que foi feito e o que ainda precisa ser feito. Se você tiver departamentos separados de desenvolvimento e garantia de qualidade, um RTM pode ser crucial para mostrar ao controle de qualidade quais testes precisam ser executados e permitir que os desenvolvedores localizem e corrijam os problemas identificados pelo controle de qualidade. Existem ainda alguns desafios que são comuns dentro da rastreabilidade de requisitos, se tudo isso parece que a rastreabilidade é muito difícil, não tema. Embora existam alguns desafios para a rastreabilidade de requisitos, também existem muitos modelos e ferramentas que você pode usar para simplificar o processo. Por enquanto, vamos enfrentar alguns dos desafios para que você possa ver como eles podem ser superados.
6- **Pontos de vista organizacionais diferentes:** Nem todos têm o mesmo entendimento de porquê e como a rastreabilidade deve ser realizada. As partes interessadas em cargos de patrocinador ou de gerenciamento só podem visualizar a rastreabilidade de requisitos de uma perspectiva padrão ou regulatória. Eles o veem como um "*must-have*", mas podem não entender os benefícios adicionais da rastreabilidade de requisitos da maneira que um engenheiro de projeto ou de

sistemas entenderia. Uma maneira de enfrentar esse obstáculo é educar as partes interessadas sobre o que pode ser alcançado se a rastreabilidade de ponta a ponta for alcançada. Compartilhe este artigo com todos os envolvidos em seu processo de desenvolvimento para que tenham uma compreensão básica de porque a rastreabilidade de requisitos é uma parte essencial do gerenciamento de requisitos, além de simplesmente saber que precisam dela para cobrir suas bases.

7- **Adoção em toda a organização:** Há uma variedade de razões pelas quais uma empresa pode demorar a adotar a rastreabilidade. O treinamento é um desses exemplos. No que diz respeito às partes interessadas, nem todas as equipes ou indivíduos que trabalham em um projeto sabem por que a rastreabilidade é tão crucial ou simplesmente não sabem como executar a rastreabilidade adequada corretamente. Além disso, algumas pessoas podem estar preocupadas que os dados de rastreabilidade de seus pontos de decisão possam voltar para prejudicá-los. Se isso é um desafio para sua organização, novamente, a educação é imperativa. Mas, além disso, você precisa criar uma cultura em que a rastreabilidade seja vista como parte inerente do processo de desenvolvimento. Comece criando políticas claras sobre como a organização gerencia a rastreabilidade. Em seguida, desenvolva um programa de treinamento positivo para todos os funcionários novos e existentes. Se você escolher uma ferramenta de gerenciamento de requisitos, certifique-se de que ela tenha um forte histórico de ser um software intuitivo e fácil de usar que se adapte ao seu processo - e não o contrário.

8- **Gestão de mudanças:** Ao construir produtos complexos, a mudança é inevitável. É essencial que os membros da equipe conheçam as mudanças e o escopo de seu impacto ao longo do ciclo de vida do desenvolvimento do produto. Isso significa examinar atentamente quaisquer requisitos de sistema relacionados, requisitos de *downstream* (fluxo de trabalho do projeto) e testes de verificação que possam ser afetados. A execução dessa atividade pode ser complicada e demorada com ferramentas manuais. Os riscos associados são semelhantes a não fazer nada, simplesmente porque você não pode ter certeza de ter contabilizado tudo ao lidar com documentos estáticos e erros humanos. Se você passou por esse revés em sua organização, talvez seja hora de explorar as ferramentas automatizadas de gerenciamento de requisitos que permitem a rastreabilidade em tempo real com requisitos ativos.

9- **Ferramentas de gestão inadequadas:** Algumas equipes de desenvolvimento ainda estão rastreando os relacionamentos de requisitos usando documentos do Word ou Excel e colaborando por e-mail. Uma Matriz de Rastreabilidade de Requisitos (RTM) é um exemplo de documento que rastreia manualmente os elementos do gerenciamento de requisitos, incluindo requisitos de negócios, objetivos, elementos de design e casos de teste por meio de uma planilha. As equipes inserem a lista de requisitos e preenchem os dados relacionados. A planilha é estática, mas é atualizada manualmente pela equipe ao longo do ciclo de desenvolvimento. Pode haver vantagens em usar um RTM se você estiver desenvolvendo um produto que não tenha muitos requisitos. É melhor do que não rastrear nada. Você pode até usar um modelo para criar um RTM. No entanto, se o seu produto for complexo, com

muitos requisitos, você provavelmente enfrentará muitos dos desafios discutidos. No caso de produtos complexos, um RTM não possui a funcionalidade necessária para acompanhar o ritmo das mudanças e criar um produto de qualidade no prazo exigido pelos stakeholders. Ferramentas flexíveis de gerenciamento de requisitos, como o Jama Connect, podem até capturar relacionamentos de rastreamento entre equipes e conjuntos de ferramentas, aprimorando ainda mais os benefícios da rastreabilidade.

10- **Estrutura de conformidade insuficiente:** Os setores regulamentados precisam de gerenciamento de requisitos para demonstrar conformidade com os padrões do setor. Existem maneiras específicas pelas quais revisores e reguladores devem receber envios regulatórios. Para ser aprovado em uma auditoria, você deve apresentar prova de rastreabilidade abrangente. Se a rastreabilidade abrangente não foi realizada durante todo o processo de desenvolvimento, muito tempo será dedicado à coleta das informações necessárias após o fato. Mesmo que a rastreabilidade tenha sido meticulosamente mantida em Word ou Excel, ainda haverá tempo gasto compilando-a em um formato aceitável para envio regulamentar. Enquanto isso, os concorrentes que tornam a rastreabilidade inerente ao seu processo serão os primeiros a entrar no mercado.

Existe algo que chamamos de melhores práticas de rastreabilidade de requisitos, que permitirão uma rastreabilidade mais eficiente.

**Identificadores únicos devem ser adotados para requisitos e regras de negócio**. Segundo Weigers (2003), para permitir a rastreabilidade, cada requisito deve ser rotulado de forma única e persistente para que você possa consultá-lo sem ambiguidades ao longo do projeto. Esta é uma boa prática para o analista empregar, ponto final, mas particularmente para rastreabilidade, onde cada requisito ou regra de negócios deve ter um identificador único e imutável durante todo o seu ciclo de vida.

**Uma parte responsável deve se apropriar da rastreabilidade.** Seja usando um método manual ou uma ferramenta automatizada, um analista experiente deve se apropriar do processo de rastreabilidade. A coleta e o gerenciamento de dados de rastreabilidade de requisitos devem ser responsabilidade explícita de certos indivíduos ou isso não acontecerá. Além disso, se alguém que não está familiarizado com o sistema ou com os requisitos tentar fazer atualizações, haverá muitos erros (WEIGERS, 2003).

**O analista deve praticar a consistência nas atualizações**. Isso exigirá um compromisso significativo por parte do analista. Sempre que tais mudanças ocorrem, é necessário atualizar os dados de rastreabilidade para refletir essas mudanças. Isso requer disciplina por parte de quem faz a mudança para atualizar os dados de rastreabilidade (WEIGERS, 2003).

Quando o rastreamento de todos os requisitos é simplesmente proibitivo, o analista pode ser seletivo com base no custo. Se a perspectiva de rastrear todos os requisitos for esmagadora, um analista pode optar por rastrear apenas os mais caros. Um método de lidar com o alto custo da rastreabilidade é praticar o rastreamento de requisitos baseado em valor em vez do rastreamento completo. Isso pode economizar uma quantidade significativa de esforço, concentrando as atividades de rastreabilidade nos requisitos mais importantes. Naturalmente, um pré-requisito para essa prática é priorizar de forma inteligente todos os requisitos de acordo com o tempo e os recursos envolvidos.

Uma organização deve adotar práticas consistentes no gerenciamento de requisitos, incluindo rastreabilidade. Se todas as partes interessadas e membros da equipe aderirem a uma prática de rastreabilidade, se apropriarem dela e se acostumarem com ela, isso aumentará muito as chances de sucesso de um método de rastreabilidade. Como observam Kannenberg e Saiedian (2009), talvez a melhor maneira de lidar com o problema dos diferentes pontos de vista das partes interessadas sobre rastreabilidade seja criar uma política organizacional sobre rastreabilidade para aplicar uniformemente a todos os projetos.

Blackburn, Busser e Nauman (2001) indicam uma sequência de como podemos efetivamente rastrear requisitos. Aqui está um método para efetivamente rastrear requisitos usando um RTM:

**1. Entenda os objetivos e escolha o formato**

Primeiro, decida porque você está fazendo uma matriz de rastreabilidade de requisitos e de que tipo você pode precisar. Uma matriz de rastreabilidade bidirecional é o tipo mais comum, pois pode mostrar a forma do seu projeto do começo ao fim. Se você estiver trabalhando para entender quais seções de codificação estão vinculadas a quais requisitos, talvez precise de uma matriz de rastreamento avançado. Se você está trabalhando para entender como tornar seu projeto mais eficiente, uma matriz de rastreamento reverso pode ajudá-lo a entender quais recursos você realmente precisa para atender às metas originais.

**2. Decida as categorias**

Depois de saber que tipo de matriz você vai construir e por quê, você pode selecionar as categorias apropriadas para rastrear. Cada matriz, provavelmente, incluirá uma coluna para nomes de requisitos e identificadores exclusivos, mas você também pode incluir alguns, como estes:

- Posição do requisito dentro do design funcional.
- Componentes de código que resolvem o design.
- Procedimento de teste para esse requisito.
- Resultados do teste para esse requisito.
- Erros encontrados ao testar esse requisito.

**3. Criar matriz**

Em seguida, crie sua matriz com uma planilha ou programa de rastreamento de requisitos. Normalmente, você pode definir suas categorias na parte superior e preencher todas as informações necessárias a seguir. Seja consistente ao inserir seus dados, especialmente os identificadores exclusivos para os requisitos, para que você possa pesquisar componentes com facilidade e precisão, se necessário.

**4. Matriz de atualização**

Para que sua matriz seja útil enquanto sua equipe trabalha em um projeto, é importante que as informações nela contidas sejam precisas quando os membros da equipe a consultarem. À medida que você progride testando componentes ou adicionando ou removendo requisitos, uma matriz atualizada pode mostrar o progresso que você fez e o que falta fazer.

## 2.3 MUDANÇAS EM REQUISITOS

Segundo Patton (2001), a mudança é um aspecto inerente ao ciclo de desenvolvimento de software. Em ambientes da vida real, muitas vezes pode ser difícil descrever o desiderato para o ciclo de desenvolvimento de software porque as circunstâncias e os requisitos do projeto estão sempre evoluindo ou mudando. Componentes de acompanhamento, como flutuação do mercado, demandas do consumidor, cenário competitivo etc., são extremamente instrumentais para a natureza maleável dos requisitos.

Além disso, a necessidade de software altamente sofisticado está em alta, já que as organizações lutam umas contra as outras para se distinguirem em um mercado extremamente competitivo. Como resultado, o gerenciamento eficaz de mudanças de requisitos no ciclo de desenvolvimento de um software torna-se crítico para o sucesso do produto final.

As mudanças no requisito podem começar logo no ponto em que ele é iniciado e podem durar além do lançamento do produto (na forma de manutenção). O processo é, portanto, definido como um sistema de gerenciamento de requisitos em mudança durante o processo de engenharia de requisitos e desenvolvimento do sistema.

O gerenciamento de mudanças é um modelo que pode ser aplicado literalmente em qualquer lugar e em qualquer situação. Ele se enquadra como uma aplicação de um processo e ferramentas estruturados que são definidas para indicar o lado humano que teremos de mudança em uma direção correta e objetiva. Em outras palavras, é um processo de engenharia de sistema onde estabelecemos e mantemos a consistência do desempenho, funcionalidade e todos os atributos físicos do produto de acordo com os requisitos, projeto e dados operacionais ao longo do ciclo de vida.

O gerenciamento de mudanças de requisitos vai de encontro aos seus procedimentos, seus processos e os padrões que são usados durante todo o gerenciamento das mudanças nos requisitos. Portanto, requisitos são itens que estão em constante mudança. Tem uma certa preocupação para que essas mudanças ocorram durante o ciclo de desenvolvimento. Tem até uma frase típica na qual fala que as mudanças de requisitos são extremamente certeiras quanto os famosos cálculos de exatas. Isso quer dizer que é muito certo termos determinadas mudanças nos requisitos de tempos em tempos, pois é quase impossível prever das suas necessidades exatas no início (ENGHOLM, 2010).

Depois dessas explicações, podemos nos perguntar: Por que os requisitos mudam? Para responder essa pergunta, vamos definir algumas possibilidades, segundo Engholm (2010):

1- **Processo de Desenvolvimento de Requisitos Mal Definidos**: Uma das principais razões para a mudança é um processo de desenvolvimento de requisitos mal definido ou ignorado. Isso pode resultar em requisitos defeituosos, requisitos incorretos e requisitos ausentes. Quanto mais longe no ciclo de vida do desenvolvimento você for antes de descobrir esses problemas, maior será o impacto no custo e no cronograma. Os desenvolvedores encontram grandes problemas e questões com os requisitos e assim começam as mudanças. Aqueles que fazem a verificação encontram requisitos que são ambíguos e não verificáveis e, portanto, a mudança continua. Lembre-se que no minuto em que começamos a fazer mudanças, mais e mais mudanças surgem – é como abrir a caixa de Pandora – não é um bom plano começar com um conjunto ruim de requisitos.
2- **Requisitos ignorados**: Outra coisa que pode ter acontecido é que os desenvolvedores simplesmente ignoraram alguns dos requisitos originais. Ou eles não entenderam os requisitos ou os requisitos eram muito complexos ou o desenvolvedor simplesmente optou por ignorá-los. Como resultado final, os desenvolvedores projetaram o produto sem referência a um ou mais dos requisitos e agora temos que voltar e corrigir o problema por meio de alterações.
3- **Tecnologia imatura**: O outro lado da questão da tecnologia é que os projetos às vezes baseiam sua capacidade de atender às expectativas das partes interessadas (ou definir as expectativas das partes interessadas) com base em uma tecnologia que não está madura o suficiente para ser usada no momento. Fazer isso adiciona riscos ao projeto, bem como o potencial de mudança se a tecnologia não funcionar.
4- **Problemas imprevistos:** A menos que você seja um oráculo, você não pode ver o futuro para prever o desconhecido e o desconhecido. Ocorrem problemas imprevistos. Os orçamentos são cortados. As datas de entrega são antecipadas. Problemas nos testes resultam em atrasos no cronograma. Um sistema com o qual você interage (interage com) muda inesperadamente. Todas essas coisas podem resultar em mudança.

5- **Necessidade de Mudanças**: Às vezes, o "problema" que seu produto deve resolver e a necessidade básica de seu produto mudam. Se isso acontecer, todo o projeto precisa ser repensado. Todas as entregas que você preparou até o momento como base para atender à Necessidade declarada originalmente. Se a Necessidade mudar, você pode ter que refazer muito do trabalho que fez até agora para atender à nova Necessidade.
6- **Partes interessadas ausentes**: as principais partes interessadas foram deixadas de fora no início do projeto. As partes interessadas representam os requisitos. As partes interessadas ausentes resultam em requisitos ausentes.
7- **Orçamento ou cronograma excessivamente otimista**: prometer algo que não pode ser entregue com os recursos disponíveis ou no tempo alocado.
8- **Interfaces não definidas adequadamente**: Uma interface chave foi negligenciada. Sem abordar a interface, o produto não pode ser integrado ao sistema macro do qual faz parte.
9- **Todos os estágios do ciclo de vida do produto não foram abordados:** Um ciclo de vida do produto chave foi perdido (teste, verificação, validação, transporte, armazenamento, transição, atualizações, manutenção etc.). Requisitos exclusivos para abordar esse ciclo de vida precisarão ser adicionados.
10- **Necessidades de mudança**: As necessidades em mudança são muitas vezes percebidas como a principal razão para a mudança. Talvez os clientes continuem mudando de ideia – eles não sabem o que querem. Este poderia ser o caso, mas, muitas vezes, os clientes não foram desafiados ou ajudados a encontrar o que realmente queriam para começar ou a entender o que era realmente necessário. Mas, em alguns casos, as necessidades realmente mudam. As necessidades de mudança acontecem com mais frequência em projetos com um ciclo de desenvolvimento longo.

Sabemos que os requisitos mudam, em várias situações possíveis, e, então, vem a pergunta: Por que precisa mudar?

Algumas pessoas não sabem o que querem até verem. Eles dizem: "Mostre-me algumas pedras e eu direi se é a pedra certa!" Você pode ter desenvolvido algo, como uma interface GUI ou um relatório que alguém queria, e quando você entrega dizem: "não, não era isso que eu queria, não era isso que eu queria dizer. O que eu realmente quis dizer é ……". Esse é um dos problemas que temos e é por isso que encorajamos a prototipagem e a modelagem e outras coisas para que possamos permitir que as pessoas vejam as coisas enquanto escrevemos os requisitos, em vez de fazendo-os esperar até que o produto seja desenvolvido.

Algumas outras ferramentas que você pode usar para ajudar seus stakeholders a comunicar o que eles realmente esperam e precisam são casos de uso, histórias e cenários que você pode usar para desenvolver um conjunto de conceitos de sistema que atendem às expectativas de todos os stakeholders, para todos os estágios do ciclo de vida e para casos nominais e não nominais. Conforme indicação de Ryser e Glinz (1999), temos:

**Expectativas em mudança**: Outro motivo para mudança é que as expectativas das partes interessadas mudam ao longo do tempo se não forem gerenciadas. Se você não estiver constantemente voltando para essas partes interessadas e estabelecendo essas expectativas para que elas saibam o que esperar, outras pessoas podem influenciá-las e mudar suas expectativas. Então, para frustrar as expectativas em mudança, comunique, comunique, comunique para definir e redefinir as expectativas das partes interessadas e minimizar possíveis mudanças.

**Mudança de tecnologia**: Outra razão para a mudança é que a tecnologia está mudando tão rápido. Temos essa tecnologia que muda rapidamente, o que significa que todos querem o que há de melhor e mais recente. Se você tem um tempo de desenvolvimento muito longo, e muitos produtos têm (projetos espaciais, aviões, automóveis e até mesmo software de contabilidade podem levar anos para serem desenvolvidos), a tecnologia estará mudando e nós estamos tentando lutar e acompanhar. Não pedimos algumas coisas no passado porque não tínhamos a tecnologia, mas agora que a tecnologia está disponível, queremos que essas coisas sejam incluídas.

**Novas partes interessadas:** algumas partes interessadas podem não existir no início do projeto. Com o tempo, algumas partes interessadas saem e são substituídas por novas partes interessadas. Cada uma dessas novas partes interessadas poderia ver o problema e a solução desejada (estado final) de maneira diferente. Por causa disso, eles podem querer mudar ou adicionar novos requisitos.

Alguns impactos devido à mudança podem surgir e fazer com que os requisitos se tornem obsoletos ou mesmo desnecessários.

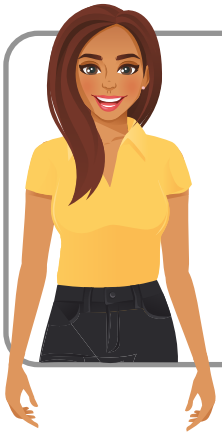
Segundo Calvo *et al*. (2002), muitos estudos foram realizados que mostram que mais de 80% dos defeitos encontrados durante a verificação foram introduzidos durante a fase de desenvolvimento de requisitos do projeto. Outros estudos mostram que uma mudança de 30% nos requisitos após a linha de base pode dobrar o custo do projeto.

Não importa que tipo de produto esteja em desenvolvimento: projetos governamentais, comerciais ou internos; hardware, software ou projetos integrados de hardware/software. Os gerentes não gostam de ouvir que o custo para desenvolver o produto será o dobro do que foi dito no início! Especialmente quando o motivo dos excessos era porque fizemos um trabalho ruim na definição dos requisitos do produto e, como resultado, tivemos muito retrabalho que elevou os custos. O marketing não quer ouvir que a data de lançamento do produto caiu. Os clientes não querem ouvir que a data de lançamento de um produto caiu (DORFMAN; THAYER, 1997).

O gerenciamento de mudanças de requisitos é definido como um processo de gerenciamento de requisitos em mudança durante o processo de engenharia de requisitos e desenvolvimento do sistema. Ter um sistema e um processo de gerenciamento de mudanças de requisitos é crucial, pois garante que as mudanças sejam feitas siste-

maticamente, as mudanças sejam documentadas e o documento de gerenciamento de mudanças de requisitos seja atualizado. Também inclui garantir que as mudanças subsequentes sejam comunicadas às partes interessadas, evitando assim falhas de comunicação. Uma análise geral do impacto da mudança pode ser feita para que as práticas de controle de mudança possam ser iniciadas (DORFMAN; THAYER, 1997).

**NOTA**



**Quando falamos de uma boa especificação de requisitos, entendemos que deve:**

- Ser clara e não ambígua.
- Serem completas.
- Serem corretas.
- Ter sua compreensão bem definida.
- Ser consistente.
- Ser concisa.
- Termos uma confiabilidade em relação aos requisitos.

Normalmente, a maioria das organizações tem um conselho de controle de mudanças que controla o gerenciamento de mudanças e toma decisões sobre a implementação das mudanças propostas.

Então, como podemos montar um plano de gerenciamento de mudanças de requisitos? A maioria das organizações geralmente define alguns elementos, como funções de gerenciamento de mudança de requisitos, painel de controle de mudança, formulário de solicitação de mudança e *changelog* (é um arquivo que compartilha uma lista cronologicamente ordenada das alterações que você realizou em seu projeto) para a superintendência de um processo eficaz de gerenciamento de mudança de requisitos.

O conselho de controle de mudanças geralmente é o corpo diretivo responsável por aprovar ou vetar solicitações de mudanças. Na maioria das vezes, o conselho de controle de mudanças também projeta uma série de etapas iterativas para garantir que a iniciativa de mudança seja bem-sucedida.

Segundo Sommerville (2007), é assim que um processo de gerenciamento de mudanças de requisitos se parece na maioria dos projetos de desenvolvimento de software:

**1. Avalie as solicitações de mudança propostas**

Os requisitos do projeto são alterados devido a vários motivos - requisitos mal definidos, requisitos modificados pelas partes interessadas ou clientes, restrições orçamentárias ou relacionadas ao cronograma ou tecnologia em evolução e tendências de mercado. A primeira etapa para alterar um conjunto de requisitos é avaliar as solicitações de mudança propostas e classificar os requisitos que precisam ser alterados como viáveis ou não viáveis, dependendo das restrições do projeto, da importância da mudança e do impacto do requisito alterado no projeto. Após comunicar a mudança, defina o escopo da solicitação de mudança em consulta e colaboração com as equipes de projeto internas e externas. O uso de um software de gerenciamento de requisitos que será capaz de agilizar o processo de gerenciamento de mudanças é muito benéfico nesta fase para evitar o aumento do escopo e restringir novas mudanças (SOMMERVILLE, 2007).

**2. Executar a mudança proposta**

Esta etapa envolve a construção de um projeto que descreve o ponto de partida, o caminho a ser percorrido e o objetivo final. Você também pode integrar os ativos a serem utilizados, a escala ou finalidade e o orçamento do plano. Isso inclui detalhar a iniciativa com etapas concretas, metas alcançáveis, benefícios, métricas e análises (SOMMERVILLE, 2007).

**3. Comunicar a mudança**

Uma vez que o impacto da mudança nos requisitos de um projeto geralmente é significativo. É imperativo que você mantenha os canais de comunicação nos dois sentidos – para cima e para baixo na hierarquia da equipe do projeto. Pequenas mudanças podem falhar no curso da comunicação e catapultar a situação para uma confusão ainda maior. Oferecer canais de comunicação diretos e claros ao longo do ciclo de vida do processo é um componente importante em todos os módulos de transição. Esses métodos promovem estruturas de responsabilidade e interação bidirecional transparente que oferecem oportunidades para expressar queixas, celebrar o que funciona e alterar efetivamente o que não funciona (SOMMERVILLE, 2007).

**4. Valide ou busque feedback**

Siga o processo de reconhecimento da ameaça percebida ou oportunidade para mudanças nos requisitos pedindo feedback. Ter perspectivas contrastantes sobre a necessidade de reforma, juntamente com visões divergentes sobre o que o processo de mudança de requisitos deve conter, pode ser uma revelação. Isso ajudará você a formular um consenso e, como todos participaram, você já terá um investimento significativo de seu pessoal principal. Como o gerenciamento de mudanças de requisitos é um processo meticuloso, não pule esta etapa de validação e aprovação mútua (SOMMERVILLE, 2007).

**5. Rejeitar ou agrupar**

Nenhuma transição ocorre sem ultrapassar alguns obstáculos. Você descobrirá que, depois de articular sua visão, formular requisitos novos e modificados, poderá encontrar resistência à mudança. Nesse ponto, você pode avaliar a resistência, colaborar e repetir a etapa 1. Você também pode seguir um sistema de alterações de requisitos em lote e implementar as propostas na próxima iteração ou sprint do ciclo de desenvolvimento do produto (SOMMERVILLE, 2007).

**6. Implementar e integrar**

Utilizar diferentes métodos para incorporar a transição no ciclo de desenvolvimento. Mantenha um sistema de monitoramento contínuo para rastrear se todos os componentes da transição estão ocorrendo corretamente ou não. Se você observar uma não conformidade, tome medidas imediatamente (SOMMERVILLE, 2007).

**7. Verifique e atualize**

A maioria dos procedimentos de mudança falha devido a uma declaração antecipada de sucesso. Implemente a mudança completamente antes de declarar uma vitória. Deixe a mudança descansar por um tempo e seja integrada por conta própria (SOMMERVILLE, 2007).

**8. Revisar e revisar**

Por mais importante que seja a integração dos requisitos em mudança no ciclo de vida de desenvolvimento do projeto, ela pode ser assustadora e até desagradável. No entanto, também é inevitável como um processo contínuo. Consequentemente, revisar o progresso do gerenciamento de mudanças de requisitos é fundamental. Idealmente, uma vez que os requisitos geralmente mudam com bastante frequência e ao longo do ciclo de vida de um projeto, um processo bem estabelecido deve ser integrado ao ciclo de vida do projeto (SOMMERVILLE, 2007).

Qual é a necessidade de um plano de gerenciamento de mudança de requisitos?

Segundo Sommerville (2007), as empresas bem-sucedidas são capazes de lidar com diferentes extensões de mudanças dramáticas e facilmente se adaptar e gerenciar o ambiente em evolução. Mudanças emergentes profundas podem ser incrivelmente perturbadoras e confusas, enquanto mudanças deliberadas e incrementais podem parecer apenas pequenas melhorias na eficiência e passar despercebidas.

Todos os tipos e níveis de mudança de requisitos precisam de alguém para liderar a carga enquanto se conecta com as partes interessadas de forma contínua. É essencial ter uma estrutura de gerenciamento de mudança clara e abrangente para ajudar a formular uma estratégia de mudança consistente, de modo a ajudar todos os envolvidos a entender o motivo da mudança, porque ela é necessária e como será seu estado futuro.

Um sistema de gerenciamento de mudanças de requisitos auxilia no gerenciamento do processo de mudança, além de oferecer controle sobre as despesas, cronograma, escala, coordenação e ativos. A estratégia de gerenciamento de mudanças também minimiza o impacto que a mudança pode ter nas empresas, funcionários, consumidores e outras partes interessadas.

Ter um sistema de gerenciamento de mudanças de requisitos é essencial, pois garante que as melhorias sejam implementadas metodicamente, as mudanças sejam registradas e o documento de gerenciamento de mudanças de requisitos registre todos os desenvolvimentos feitos até o momento. Ele também garante que as mudanças sejam transmitidas às partes interessadas relevantes, ao mesmo tempo em que garante que qualquer forma de falta de comunicação seja evitada, permitindo assim o início de práticas de controle de mudanças**.**

# RESUMO DO TÓPICO 2

**Neste tópico, você aprendeu:**

- O gerenciamento de requisitos, que é de extrema importância, pois é ele que irá capacitar todos a entender claramente as expectativas das partes interessadas e entregar com confiança um produto que foi verificado para atender aos requisitos e validar para atender às necessidades.

- O ciclo de vida dos requisitos, mostrando a abordagem sistematizada para gerenciar e manter os requisitos desde o início até a implementação final. Neste processo, visualizamos que ele possui as fases de monitoramento, planejamento, análise, gerenciamento e comunicação dos requisitos organizacionais. Com tudo isso, percebemos que os requisitos são essenciais para qualquer empresa, a fim de tomar decisões informadas e permanecer competitivas. Ao usar um processo de gerenciamento do ciclo de vida dos requisitos (LCM), você pode garantir que seus requisitos sejam gerenciados e mantidos adequadamente durante todo o ciclo de vida.

- O conceito de rastreabilidade de requisitos, que se refere à capacidade de descrever e acompanhar a vida de um requisito, tanto para frente quanto para trás (ou seja, desde suas origens, passando por seu desenvolvimento e especificação, até sua implantação e uso subsequente, e por todos os períodos de refinamento contínuo e iteração em qualquer uma dessas fases), ajudando cada vez mais os processos de software.

- O quanto as mudanças no requisito podem ser significativas no gerenciamento dele, pois vão começar logo no ponto em que o requisito é iniciado e podem durar além do lançamento do produto (na forma de manutenção). O processo é, portanto, definido como um sistema de gerenciamento de requisitos em mudança durante o processo de engenharia de requisitos e desenvolvimento do sistema.

# AUTOATIVIDADE

1 Depois de identificar e rastrear seus requisitos, você precisa garantir que eles sejam mantidos durante todo o ciclo de vida. Isso inclui mantê-los atualizados, garantindo que ainda sejam relevantes e garantindo que' não tenham sido obsoletos por outros requisitos. Os requisitos devem ser mantidos com precisão para garantir que permaneçam em um formato correto e conciso, especialmente após a aprovação de quaisquer alterações. Sobre o que é necessário para que os requisitos sejam mantidos com precisão, assinale a alternativa INCORRETA:

a) ( ) Claramente nomeado e definido.
b) ( ) Facilmente disponível para as partes interessadas.
c) ( ) Gerenciar as relações entre os requisitos.
d) ( ) Gerenciar os processos de validação.

2 Uma das principais razões para a mudança é um processo de desenvolvimento de requisitos mal definido ou ignorado. Isso pode resultar em requisitos defeituosos, requisitos incorretos e requisitos ausentes. Quanto mais longe no ciclo de vida do desenvolvimento você for antes de descobrir esses problemas, maior será o impacto no custo e no cronograma. Os desenvolvedores encontram grandes problemas e questões com os requisitos e, assim, começam as mudanças. Aqueles que fazem a verificação encontram requisitos que são ambíguos e não verificáveis e, portanto, a mudança continua. Lembre-se que, no minuto em que começamos a fazer mudanças, mais e mais mudanças surgem – é como abrir a caixa de Pandora. Não é um bom plano começar com um conjunto ruim de requisitos. Sobre as possibilidades para os requisitos mudarem, assinale a alternativa CORRETA:

a) ( ) Processo de Desenvolvimento de Requisitos Mal Definidos.
b) ( ) Requisitos ignorados.
c) ( ) Tecnologia imatura.
d) ( ) Todos os estágios do ciclo de vida do produto não foram abordados.

3 O conselho de controle de mudanças geralmente é o corpo diretivo responsável por aprovar ou vetar solicitações de mudanças. Na maioria das vezes, o conselho de controle de mudanças também projeta uma série de etapas iterativas para garantir que a iniciativa de mudança seja bem-sucedida. Sobre como um processo de gerenciamento de mudanças de requisitos deve se parecer na maioria dos projetos de desenvolvimento de software, classifique V para as sentenças verdadeiras e F para as falsas:

( ) Avaliar as solicitações de mudança propostas.
( ) Implementar e desmembrar.
( ) Validar ou buscar feedback.

Assinale a alternativa que apresenta a sequência CORRETA:
a) ( ) V – F – F.
b) ( ) V – F – V.
c) ( ) F – V – F.
d) ( ) F – F – V.

4 As empresas bem-sucedidas são capazes de lidar com diferentes extensões de mudanças dramáticas e facilmente se adaptar e gerenciar o ambiente em evolução. Mudanças emergentes profundas podem ser incrivelmente perturbadoras e confusas, enquanto mudanças deliberadas e incrementais podem parecer apenas pequenas melhorias na eficiência e passar despercebidas. Todos os tipos e níveis de mudança de requisitos precisam de alguém para liderar a carga enquanto se conecta com as partes interessadas de forma contínua. É essencial ter uma estrutura de gerenciamento de mudança clara e abrangente para ajudar a formular uma estratégia de mudança consistente, de modo a ajudar todos os envolvidos a entender o motivo da mudança, porque ela é necessária e como será seu estado futuro. Além dessas situações apresentadas no texto, quais outras necessidades podemos ter em um plano de gerenciamento de mudança de requisitos?

5 Uma organização deve adotar práticas consistentes no gerenciamento de requisitos, incluindo rastreabilidade. Se todas as partes interessadas e membros da equipe aderirem a uma prática de rastreabilidade, se apropriarem dela e se acostumarem com ela, isso aumentará muito as chances de sucesso de um método de rastreabilidade. Como observam Kannenberg e Saiedian (2009), talvez a melhor maneira de lidar com o problema dos diferentes pontos de vista das partes interessadas sobre rastreabilidade seja criar uma política organizacional sobre rastreabilidade para aplicar uniformemente a todos os projetos. Diante dessas conceituações, como podemos indicar uma efetiva rastreabilidade de requisitos utilizando o método de RTM? Justifique.

# O MERCADO DE TRABALHO DE ENGENHARIA DE REQUISITOS

## 1 INTRODUÇÃO

Acadêmico, no Tema de Aprendizagem 3, abordaremos o mercado de trabalho para engenheiros de requisitos de software. Este mercado está em constante expansão, devido ao fato de que a maioria das atividades cotidianas das pessoas estão migrando totalmente para as plataformas digitais.

Caso você tenha feito algum tipo de pesquisa para verificar o mercado de engenharia de requisitos, possivelmente já deve ter descoberto que existem vários tipos de engenheiro de software. Na maioria das vezes, os níveis de emprego são diferentes. Esses níveis são fundamentais para entender a habilidade e a experiência de cada funcionário. Para poder verificar e ajudar nessa análise, o referencial é somente baseado em três níveis, que são: iniciantes, plenos e seniores.

No caso de profissionais recém-formados, a grande maioria procura ocupar as vagas em áreas de diferentes segmentos, por exemplo: desenvolvedores de sistemas, gerenciamento de bancos de dados e infraestrutura de sistemas, áreas que são vistas com uma maior empolgação por parte dessas pessoas. Com isso, é nítido o aumento de empresas procurando profissionais ligados à engenharia de software e requisitos.

Assim, entendemos que esse profissional, engenheiro de software/requisitos, pode atuar em várias determinações. Veremos as suas responsabilidades a seguir.

**INTERESSANTE**

O analista de requisitos é a pessoa responsável por atuar com análise de requisitos funcionais, não-funcionais, ou mesmo de usuários, dos clientes e também na análise de negócio. Este profissional é responsável por fazer a especificação e a análise de sistemas e, na maioria dos casos, faz algumas especificações funcionais, objetivando o desenvolvimento de software.

Conforme Zahran (1998), a engenharia de requisitos (ER) também é muito afetada pela mudança de perspectiva. Tradicionalmente, RE é descrito como "quais serviços um produto deve fornecer e sob quais restrições ele deve operar", e na perspectiva sob medida RE consiste no processo sistemático de elicitar, entender, analisar, documentar e gerenciar requisitos ao longo do ciclo de vida de um sistema. O foco aqui está na instância de desenvolvimento (projeto) em si, ou seja, o esforço de ER é iniciado pelo projeto e parte de, por exemplo um pré-estudo ou focado nos estágios iniciais de desenvolvimento.

Sommerville (2007) orienta que, em contraste, a engenharia de requisitos orientada para o mercado (MDRE) tem um fluxo contínuo de requisitos e o esforço de engenharia de requisitos não se limita a uma instância de desenvolvimento, mas uma parte do gerenciamento de produto como um todo. Nesse ambiente, os requisitos vêm de várias fontes, tanto internas (por exemplo, desenvolvedores, marketing, vendas, equipe de suporte, relatórios de bugs etc.).

Isso pode dar origem a quantidades muito grandes de requisitos, e todos eles precisam ser capturados, especificados, analisados e gerenciados continuamente à medida que o produto evolui ao longo do tempo através dos lançamentos. As atividades de seleção de requisitos de MDRE (planejamento de liberação) são centrais, ou seja, a decisão sobre quais clientes obtêm quais recursos e qualidade em quais ponto no tempo. Isso torna a precisão do planejamento de lançamento um dos principais determinantes do sucesso de um produto. No MDRE o foco está no produto e nos requisitos, e os esforços de desenvolvimento (projetos) são iniciados por requisitos (SOMMERVILLE, 2007).

Para Davis (1993), a importância de ter um processo de ER adequado em vigor que produza requisitos suficientemente bons pode ser considerada crucial para o desenvolvimento bem-sucedido de produtos, seja em um esforço de desenvolvimento sob medida ou orientado para o mercado. No entanto, há indicações claras de que a engenharia de requisitos está faltando na indústria, uma vez que as inadequações nos requisitos são um dos principais determinantes do fracasso do projeto. Muitos dos desafios são relevantes tanto para RE quanto para MDRE sob medida. Por exemplo, problemas (qualidade inadequada) nos requisitos filtram para projetar e implantar. Davis (1993) publicou resultados indicando que poderia ser até 200 vezes mais caro detectar e reparar defeitos durante a fase de manutenção de um sistema em comparação com a engenharia de requisitos, e várias outras fontes indicam que requisitos inadequados são a fonte principal para o fracasso do projeto. Isso é ainda agravado no caso de o MDRE como seleção inicial de requisitos ser crucial, em particular porque grandes volumes de requisitos de várias fontes correm o risco sobrecarregar as empresas. É vital que os requisitos recebidos possam ser tratados de forma estruturada, descartando os irrelevantes em um estágio inicial, gastando o mínimo de esforço possível para economizar recursos para refinar requisitos que serão realmente selecionados e alocados para o desenvolvimento de instâncias. Além do volume, as próprias necessidades são

de qualidade variável, estado de refinamento e nível de abstração. No desenvolvimento sob medida tradicional, um engenheiro de requisitos pode eliciar requisitos ativamente e, assim, esperar controlar ou pelo menos influenciar substancialmente esses aspectos. Em uma situação impulsionada pelo mercado, esse raramente é o caso. Sommerville e Sawyer (1997) dizem que a maioria dos requisitos já são declarados de uma forma ou de outra quando atingem o engenheiro de requisitos (por exemplo, um gerente de produto). O conhecimento e a experiência da organização em desenvolvimento, da qual o engenheiro de requisitos está, neste caso é fundamental para dar sentido aos requisitos à medida que são processados.

Existem muitos guias de melhores práticas de ER e estruturas de melhoria de processo de software que visam encontrar desafios/problemas que podem ser usados como base para melhorar as práticas de ER.

No entanto, a maioria deles é adaptada para se adequar a um ambiente sob medida com relacionamentos tradicionais, focados em projetos, cliente-desenvolvedor. Outro problema é que muitas estruturas de melhoria de processo de software (SPI) são, muitas vezes, grandes e volumosos demais para obter uma visão geral e, se implementado, o retorno sobre o investimento pode levar de 18 a 24 meses. Isso torna difícil para as organizações, em particular as pequenas e médias empresas (PMEs), para iniciar e realizar a avaliação e atividades de melhoria como custo e tempo são considerações cruciais. Segundo Calvo *et al*. (2002), existe a necessidade de desenvolver e avaliar as melhores práticas guias, técnicas e modelos que dão suporte aos profissionais que atuam na ambientes orientados para o mercado - assumindo os desafios/problemas únicos de MDRE em conta. Um pré-requisito para identificar esses desafios na indústria é o desenvolvimento de estruturas apropriadas de avaliação e melhoria de processos.

Segundo Basili e Green (1994), a engenharia de requisitos orientada para o mercado (MDRE) é, em muitos aspectos, muito semelhante à engenharia de requisitos (RE) tradicional sob medida. Muitos dos as práticas são as mesmas, assim como muitas das habilidades necessárias. No entanto, lá existem várias diferenças cruciais que precisam ser reconhecidas para compreender plenamente os desafios enfrentados pelos profissionais que trabalham em um ambiente orientado para o mercado.

Muitas organizações se deparam com uma mistura de tendências voltadas para o mercado e desenvolvimento de produtos sob medida. Um exemplo pode ser uma empresa que oferece produtos para um mercado, mas ao mesmo tempo realizando adaptações para clientes que desejam pagar pela confecção dos produtos. Essa situação torna necessário adotar RE sob medida para, por exemplo, projetos de adaptação, ao mesmo tempo em que possui um processo MDRE que lida com a engenharia de requisitos contínuos que rege o desenvolvimento de produtos para o mercado.

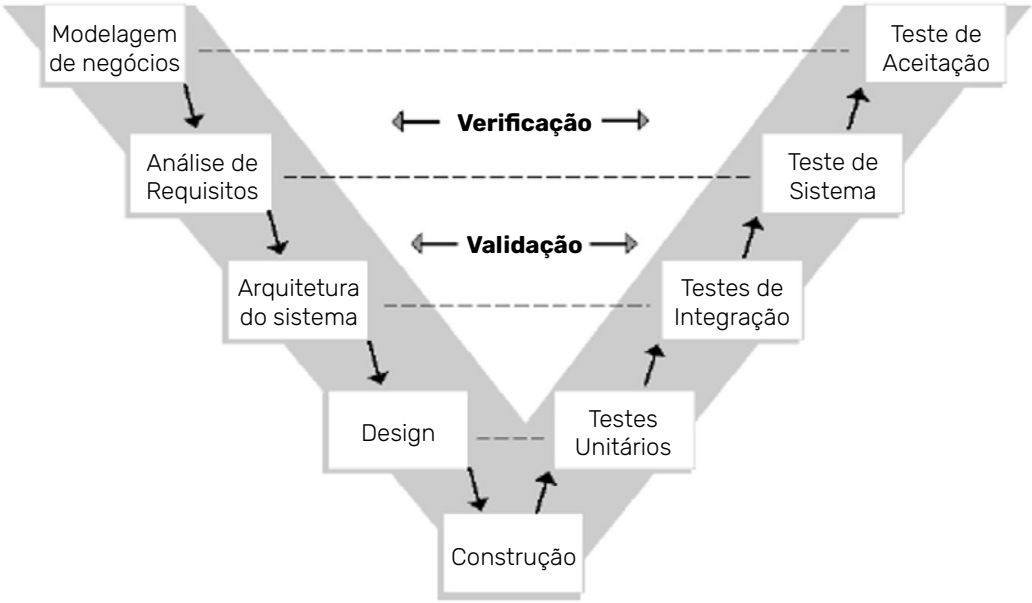
# 2 VALIDAÇÃO E VERIFICAÇÃO DE REQUISITOS DE SOFTWARE

O processo de verificação garante que o software atenda às especificações. A validação se concentra em saber se o software atende às expectativas e requisitos do usuário final. Desenvolvimento de Requisitos, Verificação de Requisitos, Validação de Requisitos, Verificação de Sistema e Validação do sistema são tarefas importantes. Este papel começa com uma definição e explicação desses termos. Então, dá duas dúzias de exemplos de falhas de sistema famosas e sugere os erros que podem ter sido cometidos.

Segundo Pressman (2006), cada requisito deve ser verificado e validado para garantir que seja o requisito correto. Isso garante que os requisitos atendam ao objetivo geral do sistema e a todas as necessidades das partes interessadas. A verificação e a validação devem ser feitas continuamente ao longo do desenvolvimento de requisitos em todos os níveis e como parte das atividades de linha de base e revisadas durante as Revisões de Requisitos do Sistema (SRR) .

A figura 4, mostra exatamente a relação entre verificação e validação de requisitos:

Figura 4 – Verificação x Validação




Fonte: adaptado de Conventry (2015)

Como os termos de verificação e validação são muitas vezes confusos, vamos examinar as seguintes definições (CONVENTRY,2015):

- **Verificação de requisitos**: prova de que cada requisito foi satisfeito. A verificação pode ser feita por argumento lógico, inspeção, modelagem, simulação, análise, revisão especializada, teste ou demonstração.
- **Requisitos de validação**: garantir que (1) o conjunto dos requisitos é correto, completo e consistente, (2) pode ser criado um modelo que satisfaça os requisitos e (3) uma solução do mundo real pode ser construída e testada para provar que satisfaz os requisitos. Se a Engenharia de Sistemas descobrir que o cliente solicitou uma máquina de movimento perpétuo, o projeto deve ser interrompido.
- **Verificando um sistema**: Construindo o sistema corretamente: garantindo que o sistema esteja em conformidade com o sistema requisitos e está em conformidade com o seu design.
- **Validando um sistema**: Construindo o sistema certo: certificando-se de que o sistema faz o que deve fazer em seu ambiente pretendido. A validação determina a exatidão e integridade do produto final, e garante que o sistema irá satisfazer as necessidades reais das partes interessadas.

Há sobreposição entre a verificação do sistema e verificação de requisitos. A verificação do sistema garante que o sistema está em conformidade com seu projeto e com os requisitos do sistema. A verificação de requisitos garante que os requisitos do sistema sejam atendidos e que sejam verificados os requisitos técnicos, derivados e de produto. Portanto, verifique os requisitos do sistema é comum a ambos os processos.

Também há sobreposição entre a validação de requisitos e a validação do sistema. Validando o nível superior dos requisitos do sistema é semelhante à validação do sistema, mas validar requisitos de baixo nível é bem diferente da validação do sistema.

Muitos engenheiros de sistemas e engenheiros de software usam as palavras verificação e validação no oposto da moda. Portanto, é necessário concordar com as definições de verificação e validação.

As áreas de processo de Verificação (VER) e Validação (VAL) na Integração do Modelo de Maturidade de Capacidade (CMMI) falam de, respectivamente, verificar requisitos e validação do sistema. A validação dos requisitos é coberta no Desenvolvimento de Requisitos (RD).

## 2.1 O QUE É VERIFICAÇÃO DE REQUISITOS?

Se você usou verificação e validação de forma intercambiável no passado, uma das coisas mais importantes a observar é a ordem. A verificação do software vem primeiro, seguida pela validação. O que está envolvido com cada um? Vamos mergulhar na verificação primeiro.

Os testes de verificação conferem se o programa foi construído de acordo com os requisitos declarados. O processo de verificação inclui atividades como revisar o código e fazer orientações e inspeções.

Segundo Pfleeger (2004), requisitos ausentes ou requisitos inválidos podem ser descobertos durante esta fase, o que pode minimizar o risco de retrabalho e o custo associado a excessos. É muito mais eficaz corrigir um pequeno bug antecipadamente do que no futuro, quando centenas de linhas de código devem ser identificadas e corrigidas.

Por exemplo, imagine que você está dirigindo para um novo destino. Você pode inserir esse destino em seu GPS, que fornece direções e o número da saída da rodovia. Se você está procurando a saída 10 e acabou de passar pela saída 1, você rapidamente sabe que tem mais nove saídas pela frente. Isso é semelhante à frase de verificação, pois o uso do GPS permite que você verifique seu caminho existente em relação às direções.

Outro exemplo é inserir uma fórmula em uma planilha. Depois de inserir algumas linhas de dados, você pode verificar a fórmula e certificar-se de que está funcionando. O processo de verificação é o mesmo, pois permite que você faça uma verificação rápida antes de se aprofundar no processo de desenvolvimento do produto.

Cada requisito deve ser verificado por argumento lógico, inspeção, modelagem, simulação, análise, revisão especializada, teste ou demonstração. Aqui estão algumas breves definições de dicionário para esses termos (MATIELLO-FRANCISCO *et al*., 2006):

- **Argumento lógico**: uma série de deduções lógicas.
- **Inspeção**: examinar cuidadosa e criticamente, especialmente em busca de falhas.
- **Modelagem**: uma representação simplificada de algum aspecto de um sistema.
- **Simulação**: execução de um modelo, geralmente com programa de computador.
- **Análise**: uma série de deduções lógicas usando matemática e modelos.
- **Revisão pericial**: um exame dos requisitos por um painel de especialistas.
- **Teste**: aplicando entradas e medindo saídas sob condições controladas (por exemplo, um ambiente de laboratório).
- **Demonstração**: mostrar por experiência ou prática aplicação (por exemplo, um voo ou teste de estrada). Algumas fontes dizem que a demonstração é menos quantitativa do que testar.
- A modelagem pode ser uma técnica de verificação independente, mas frequentemente os resultados da modelagem são usados para dar suporte outras técnicas.

- **Exemplo de verificação de requisitos 1**: A probabilidade de receber um bit incorreto no canal de telecomunicações deve ser menor que 0,001. Esse requisito pode ser verificado por testes de laboratório ou demonstração em um sistema real.
- **Exemplo de verificação de requisitos 2**: A probabilidade de perda de vidas em uma missão tripulada a Marte deve ser menor que 0,001. Este certamente é um requisito razoável, mas não pode ser verificado por meio de teste. Isto pode ser possível verificar esse requisito com análise e simulação.
- **Exemplo de verificação de requisitos 3**: A probabilidade de o sistema ser cancelado por políticos deve ser menor que 0,001. Embora isso possa ser um bom requisito, não pode ser verificado com teste de engenharia normal ou análise. Pode ser possível verificar esse requisito com argumentos lógicos.

## 2.2 O QUE É VALIDAÇÃO DE REQUISITOS?

A resposta a essa pergunta pode variar dependendo do setor em que você atua e do tipo de requisito que está sendo validado.

Em geral, depois de ter qualificado que tipo de requisito está sendo validado e concluído a verificação, é hora de concluir o teste de validação, que confirma a precisão dos requisitos. Ele garante que os requisitos atinjam os objetivos de negócios, atendam às necessidades de quaisquer partes interessadas relevantes e sejam claramente compreendidos pelos desenvolvedores.

Pfleeger (2004) indica que validar requisitos significa garantir que: (1) o conjunto dos requisitos é correto, completo e consistente; (2) um modelo que satisfaça os requisitos pode ser criado; e (3) uma solução do mundo real pode ser construída e testada para provar que satisfaz os requisitos. Se os requisitos especificarem um sistema que reduz a entropia sem gasto de energia, então os requisitos não são válidos e o projeto deve ser interrompido.

Segundo Matiello-Francisco *et al.* (2006), a validação é uma etapa crítica para encontrar requisitos ausentes e garantir que os requisitos tenham uma variedade de características importantes. A validação de software aborda o seguinte:

- Descreve corretamente a necessidade do usuário final.
- Tem apenas um significado exato.
- Pode ser modificado conforme necessário.
- Documenta os atributos que os clientes realmente precisam.
- Facilmente vinculado aos requisitos do sistema, como projetos, códigos e testes.

A validação não está focada no caminho que você percorreu para chegar ao destino, mas sim em saber se você atingiu o alvo. Por exemplo, considere o último exemplo de uma pessoa viajando em um carro e rastreando pontos de referência, como números de saída. Digamos que o objetivo seja chegar a uma trilha de caminhada. Algumas perguntas podem ser feitas quando você chegar.

- A trilha de caminhada parece o esperado?
- Posso ver uma trilha marcada e um sinal de trilha?
- O local atende às minhas expectativas?

A validação é focada nos mesmos tipos de perguntas. Não está preocupado em como você chegou lá, mas em como você chegou ao local correto.

Se você estiver projetando uma planilha, conforme discutimos antes, você verificou se a fórmula funcionou durante o processo de verificação. Durante a validação, você garante que o produto final (a planilha) atenda às necessidades do usuário.

Por exemplo, usando o exemplo da planilha, talvez tenhamos determinado que vamos usar uma planilha para reduzir o tempo que levamos para "x" ou reduzir os erros que vemos quando "y", então implementamos uma planilha com um monte de funções, as funções definidas por nossos requisitos. A verificação pergunta se a planilha funciona de acordo com os requisitos ("construímos corretamente"); A validação pergunta se a planilha realmente cumpre as metas relacionadas a x e y ("construímos a coisa certa").

Mesmo quando um produto é totalmente verificado – ele ainda pode não atender às necessidades do usuário, ou seja, falha na validação. Aqui está um exemplo de um conjunto de requisitos inválidos para um controlador de aquecedor elétrico de água.

If 70° < Temperatura < 100°, a saída é 3000 Watts.
If 100° < Temperatura < 130°, a saída é 2000 Watts.
If 120° < Temperatura < 150°, a saída é 1000 Watts.
If 150° < Temperatura, a saída é 0 Watts.

Este conjunto de requisitos está incompleto, o que deveria acontecer se a temperatura < 70°? Este conjunto de requisitos é inconsistente, o que deveria acontecer se Temperatura = 125°? Esses requisitos estão incorretos, porque as unidades não são dados. Essas temperaturas são em graus Fahrenheit ou Centígrados?

Claro, você nunca poderia provar que um conjunto de requisito estava completo, e talvez fosse muito caro para fazer isso, mas estamos sugerindo que, muitas vezes, devido à estrutura do conjunto de requisitos, você pode procurar incompletude.

Segundo Barreto (2006), os defeitos de validação de requisitos detectáveis incluem (1) conjuntos incompletos ou inconsistentes de requisitos ou casos de uso, (2) requisitos que não rastreiam ao nível superior requisitos a declaração de visão ou o conceito de Operação (CONOPS), e (3) casos de teste que não trace para cenários (casos de uso).

Nas inspeções, o papel do testador deve receber uma responsabilidade adicional, validação de requisitos. O testador deve ler a Visão e o CONOPS e procurar especificamente por defeitos de validação de requisitos, como esses.

## 2.3 VERIFICAÇÃO E VALIDAÇÃO DO SISTEMA

Segundo Abreu, Farias e Albuquerque (2009), uma função de *Stonehenge* em *Salisbury Plain* na Inglaterra pode ter servido como um calendário para indicar os melhores dias para plantar. Isso pode ter sido o primeiro calendário, e sugere a invenção do conceito de tempo. Inspirado por uma visita a *Stonehenge*, Terry Bahill construiu uma visão do pôr do sol do Equinócio de Outono no telhado de sua casa na cidade de Tucson. O autor buscava documentos de verificação e validação para este calendário solar, embora devesse ter preocupado com isso antes que o hardware fosse construído. O sistema falhou na validação. Ele precisava de um detector do equinócio vernal, não de um detector de equinócio de outono. Os antigos em Tucson precisavam de um detector de solstício de verão, porque toda a chuva vem em julho e agosto. A validação do sistema requer consideração do ambiente em que o sistema irá operar (BAHILL; HENDERSON, 2005).

Em 3000 a.C., os engenheiros de Stonehenge puderam verificaram o sistema marcando o pôr do sol a cada dia. Os solstícios são os mais distantes ao norte e ao sul (aproximadamente). Os equinócios estão a meio caminho entre os solstícios e são diretamente leste e oeste. No século XXI, os moradores de Tucson puderam verificar o sistema consultando um calendário ou almanaque de um fazendeiro e observando o pôr do sol através desta vista no equinócio de outono no próximo ano. Se o pôr do sol está à vista no dia do equinócio de outono, então o sistema foi construído certo.

Artefatos de validação do sistema que podem ser coletados continuamente ao longo do ciclo de vida incluem resultados de modelagem e simulação e o número de operações cenários (casos de uso) modelados.

Segundo Basili e Green (1994), os defeitos de validação do sistema detectáveis incluem (1) sensibilidade excessiva do modelo a um parâmetro específico ou requisito, (2) incompatibilidades entre o modelo/simulação e o sistema real, e (3) projetos ruins.

Nas inspeções, o papel do testador deve receber uma responsabilidade adicional, validação do sistema. Testador devem ler a Visão e o CONOPS e especificamente procure por artefatos e defeitos de validação do sistema, como esses.

Um aspecto muito importante da validação do sistema é que ocorre durante todo o ciclo de vida do sistema. Você não deve esperar pelo primeiro protótipo antes de começar atividades de validação.

Ainda conforme Basili e Green (1994), as atividades de verificação e validação do sistema devem começar na fase de proposta. Verificação e validação são processos contínuos que são feitos ao longo do ciclo de vida de desenvolvimento do sistema. Portanto, a maioria dessas atividades será interna à empresa. No entanto, também é importante ter recursos externos de verificação e validação. Isso poderia ser feito por uma divisão ou empresa independente. A verificação e validação externas devem envolver o uso do sistema pelo cliente e o usuário final na intenção do sistema de ambiente operacional. Este tipo de verificação e validação externa não seria feito ao longo do ciclo de desenvolvimento. Não ocorreria até pelo menos um protótipo estava disponível para teste. Isso é uma das razões pelas quais a comunidade de software enfatiza a importância de desenvolver protótipos no início do processo de desenvolvimento.

Para engenheiros de sistema, ser capaz de rastrear relacionamentos entre tipos de dados é essencial. No entanto, pode haver um problema com vários níveis de requisitos, especificações e artefatos de verificação, todos com seu próprio conjunto de partes interessadas que executam uma variedade de tarefas.

A solução de software certa pode simplificar situações complicadas e permitir que você adicione rastreabilidade dos dados. Você pode analisar "quem, o quê, onde e por quê" das possíveis mudanças e garantir que os dados essenciais não sejam negligenciados. Procure uma solução de software que faça o seguinte:

Conecta casos de teste de uma declaração de problema aos seus requisitos e design. Se você não tem a capacidade de fazer isso, não pode ter certeza de que não esqueceu algo crítico.

Conecta os requisitos do sistema aos requisitos do negócio e das partes interessadas. Se você perder uma conexão crítica, corre o risco de despesas não planejadas que podem ter um efeito cascata e criar lentidão nos lançamentos de produtos, enfraquecer a confiança das partes interessadas e afetar adversamente os resultados financeiros.

É fundamental relacionar os requisitos de nível inferior aos requisitos de nível superior para garantir que os componentes e subcomponentes se unam em um sistema funcional. Erros nessa área podem levar a custos extras, pois você trabalha duro para juntar as peças e implementar alterações posteriormente no processo de desenvolvimento do produto.

Uma solução de software é uma ferramenta crítica para ajudá-lo a gerenciar o processo de verificação e validação e garantir que todas as atividades de engenharia estejam conectadas durante todo o ciclo de vida do sistema. É fundamental capturar toda a comunicação no contexto e reunir as partes interessadas em um só lugar para uma visão abrangente e em tempo real do que as equipes estão construindo e por quê.

Podemos também começar a pensar no futuro das verificações e validações de requisitos. O teste pode ser uma das partes mais caras do desenvolvimento do produto, se não for planejado adequadamente. É importante incorporar verificação e validação para garantir economia de custos e um produto de alta qualidade. No final, se o produto não atender aos objetivos originais, tempo, dinheiro e esforço terão sido desperdiçados.

Felizmente, as empresas podem lançar produtos no mercado mais rapidamente quando as pessoas e os dados permanecem sincronizados com as atividades e entregas de desenvolvimento de produtos. O uso de ferramentas inovadoras de software de gerenciamento de requisitos pode facilmente reduzir o tempo desde a concepção até a criação de valor e desempenho.

# 3 FERRAMENTAS PARA ENGENHARIA DE REQUSITOS

Uma grande parte do papel de um gerente de engenharia é criar estratégias eficazes, implementar planos de projeto e garantir que tudo e todos estejam no caminho certo. É aqui que entra o gerenciamento de requisitos. Com a natureza complexa dos projetos e várias equipes para gerenciar, os gerentes de engenharia podem se tornar um pesadelo para controlar tudo manualmente.

Felizmente, existem algumas ferramentas e softwares para ajudar no processo. Essas ferramentas, chamadas coletivamente de ferramentas de engenharia de requisitos, podem facilitar muito a vida dos gerentes de engenharia. Dada a natureza exigente dos locais de trabalho modernos, é essencial que os gerentes de engenharia aprendam essas ferramentas, agregando valor ao seu trabalho.

A engenharia de requisitos é a abordagem disciplinada e sistemática para elicitar, especificar, analisar, confirmar, validar e gerenciar requisitos enquanto considerando as necessidades do usuário, técnicas, econômicas e de negócios e objetivos. Abrange todo o ciclo de vida, muitas vezes envolvendo equipes distribuídas e cadeias de suprimentos. As ferramentas facilitam a consistência e a eficiência no gerenciamento requisitos. Descobrir qual ferramenta é adequada para determinadas necessidades não é fácil.

Segundo Machado (2011), as ferramentas de engenharia de requisitos (RE) estão evoluindo rapidamente. A demanda por flexibilidade, magra e desenvolvimento ágil, colaboração mundial e ecossistemas avançados de software e sistemas está mudando a forma como gerenciamos os requisitos. Por exemplo, equipes ágeis são

menos centradas em documentos e mais orientado a código - eles esperam breves requisitos diretamente relacionados a mudanças de código - portanto, sua ferramenta de RE deve ser leve. Por outro lado, equipes de desenvolvimento distribuídas com expectativas de alta qualidade exigem especificações fortes, requisitos abrangentes e rastreabilidade durante todo o ciclo de vida. Sua ferramenta RE deve garantir o serviço por gerações de software. As ferramentas de ER estão se adaptando a essas demandas com mudanças em seu design e arquitetura. Tradicionalmente, RE e as ferramentas são proprietárias e bem mantidas por seus fornecedores; eles são frequentemente orientados para diferentes ambientes e nichos de mercado (por exemplo, automotivo, médico e de defesa), processos de desenvolvimento (por exemplo, desenvolvimento ágil, gerenciamento de produtos e prototipagem) ou configurações de utilização (para o exemplo, desenvolvimento de software local versus global). Razão suficiente para avaliar ferramentas e tecnologias de ER com diferentes casos de uso.

Quando falamos de gerenciamento de requisitos, é importante destacarmos algumas ferramentas que irão fazer essa gestão e ajudar em todo o processo.

Vamos citar a seguir algumas ferramentas que irão facilitar bastante o trabalho dos profissionais:

**Jama Software (https://www.jamasoftware.com)**

Os usuários podem mapear processos e relacionamentos entre eles, bem como rastrear riscos e requisitos. A Jama Software fornece uma plataforma para gerenciamento de requisitos, riscos e testes. As equipes que constroem produtos, sistemas e softwares complexos usam o Jama Connect para melhorar os tempos de ciclo, aumentar a qualidade, reduzir o retrabalho e minimizar o esforço para provar a conformidade.

Essa ferramenta de nível empresarial oferece a capacidade de capturar e comunicar com precisão requisitos, metas, progresso e interdependências em todo o processo de desenvolvimento. Ele permite que você alinhe as partes interessadas no desenvolvimento de hardware, firmware e software; melhorar a eficiência; gerenciar mudanças; e comprovar o cumprimento.

Nos recursos do Jama Connect, você encontrará rastreabilidade ao vivo de ponta a ponta de pessoas, dados e processos, bem como análises de impacto, revisões e aprovações em tempo real com a colaboração da equipe. Para coleta de requisitos, você pode definir, organizar e executar planos de teste baseados em requisitos e casos de teste para garantir qualidade e conformidade.

Todos esses recursos são agrupados em um software que oferece flexibilidade para dar suporte a várias disciplinas de engenharia e metodologias de desenvolvimento. Combine isso com uma interface intuitiva baseada em navegador e você terá um software abrangente que ainda é fácil de navegar.

Você pode integrar os requisitos e planos de teste do Jama Connect com ALM, PLM, QA e MBSE para garantir total rastreabilidade, visibilidade e colaboração em todo o ciclo de vida.

Com integrações de API e hubs de integração de terceiros, as equipes podem trabalhar em ferramentas familiares enquanto contribuem para projetos no Jama Connect. As licenças do Jama Connect estão disponíveis por usuário e flutuante por meio de um modelo de assinatura anual.

Além de registrar os requisitos do produto, os usuários podem acompanhar a rastreabilidade, os defeitos e a verificação.

**Virsure (https://visuresolutions.com/pt/)**

O Visure fornece uma solução personalizada e fácil de usar para gerenciamento de requisitos, riscos e testes. É uma ferramenta obrigatória para equipes que constroem produtos, sistemas e softwares complexos, que exigem rastreabilidade de ponta a ponta desde a concepção até o teste e a implantação, juntamente com a conformidade com a certificação padrão. É uma das plataformas de RM mais flexíveis e personalizáveis, facilitando a colaboração das partes interessadas em requisitos simples ou complexos.

As empresas que usam ativamente o Visure reivindicam um impacto claro com entregas de projetos no prazo, conformidade do projeto e redução nos custos de desenvolvimento e tempos de ciclo.

O Visure oferece suporte a um dos melhores recursos para exportação e importação do MS Office e integra-se perfeitamente com IBM DOORS, JIRA, Sparx EA, TFS, ReqIF, MATLAB, VectorCAST e outras ferramentas importantes para equipes de engenharia de sistemas. Ele também oferece suporte a uma ampla variedade de tipos e metodologias de projeto, incluindo modelos ágeis, em cascata e V.

Outros recursos essenciais da plataforma incluem: rastreabilidade de ponta a ponta, desde a concepção até o teste e a implantação, até o código-fonte como Java, C, C++; representação gráfica de seu fluxo de trabalho (modelo de dados) para impor e proteger uma rastreabilidade total; análise de impacto com um clique e gerenciamento de testes; e análise de qualidade orientada por IA para verificar automaticamente a qualidade de seus requisitos.

Esta ferramenta RM usa processamento de linguagem natural para aplicar as melhores táticas. Você pode criar componentes, requisitos e casos de teste reutilizáveis para usar em equipes e projetos. Você pode aproveitar a integração total do Word e do Excel para obter facilmente os requisitos de usuários que não são do Visure.

Eles fornecem um suporte premium e modelos prontos para uso adaptados a diferentes padrões da indústria, como ISO 26262, IEC 62304, IEC 61508, CENELEC 50128, DO-178B/C, FMEA, SPICE e CMMI.

E, por último, mas não menos importante, oferece uma das plataformas de RM corporativas mais acessíveis do mercado atualmente.

**ReqSuite (https://www.osseno.com/en/requirements-management-tool/)**

O ReqSuite® RM da OSSENO Software é uma solução muito poderosa para gerenciar requisitos, casos de teste e outros artefatos conceituais ao longo do ciclo de desenvolvimento.

O software fornece um rico kit de ferramentas com poderosas opções de personalização, bem como a capacidade de gerenciar, rastrear, analisar, aprovar, revisar, exportar, importar e reutilizar requisitos de forma colaborativa. Além disso, sua facilidade de uso e configuração simples, bem como suporte premium gratuito, significa que você pode obter uma solução personalizada em execução rápida.

A ferramenta também inclui assistência baseada em IA, que ajuda as empresas com controle automático de qualidade e correspondência de requisitos. O ReqSuite® RM também está em conformidade com os padrões ISO 26262, IEC 60812, ISO 13485, ISO 14971, FDA 820.30 e DO176C.

As integrações incluem sincronização bidirecional com várias ferramentas de terceiros, incluindo Jira, Azure DevOps, Enterprise Architect, GitLab, Redmine, TestRail, Word, Excel e ReqIF - e sua WebAPI permite que você se conecte a qualquer outro sistema.

Você pode obter este software como uma plataforma local ou hospedada na nuvem a partir de € 75,00 por licença nomeada e testá-lo por 14 dias completos, pois uma versão de avaliação gratuita está disponível (somente para clientes em potencial da Europa e América do Norte).

**Codebeamer (https://codebeamer.com/cb/login.spr)**

Melhor para integrações de API e prontas para uso. Identifique, gerencie e rastreie os requisitos em todo o ciclo de vida com o codebeamer. Reduza custos, riscos e prazos de entrega no desenvolvimento de produtos.

O codebeamer da Intland Software é uma plataforma completa de gerenciamento de ciclo de vida de aplicativos de ponta a ponta com fortes recursos de gerenciamento de requisitos. A ferramenta ajuda a gerenciar a complexidade desde os requisitos até a liberação na entrega de produtos de software e sistemas de sistemas críticos ou tradicionais para segurança.

Para desenvolvedores de produtos e softwares avançados, essa plataforma aberta estende as funcionalidades do ALM com configuração de linha de produtos e fornece configurabilidade exclusiva para processos complexos. Alinhe pessoas, fluxos de trabalho e ferramentas nos fluxos paralelos de entrega de produtos, capacitando as equipes com uma plataforma flexível, totalmente integrada e altamente colaborativa.

O codebeamer é uma ótima ferramenta para apoiar engenheiros que procuram rastreabilidade, transparência e colaboração eficiente no gerenciamento de requisitos. A plataforma fornece rastreabilidade contínua para ajudar a rastrear as interdependências e o impacto das mudanças ao longo do ciclo de vida. O teste baseado em requisitos é suportado imediatamente. Use o codebeamer para identificar, gerenciar e rastrear requisitos em todo o ciclo de vida, a fim de reduzir custos, riscos e prazos de entrega no desenvolvimento de produtos.

Opções de relatórios personalizados, controle de alterações automatizado e gerenciamento de aprovação simples ajudam a simplificar a colaboração em ambientes de desenvolvimento de alta velocidade. Para desenvolvedores de produtos regulamentados, os modelos e serviços de domínio pré-configurados da Intland ajudam a reduzir o esforço e os custos de conformidade regulatória.

A ferramenta se integra com Jira, Microsoft Word, Excel, IBM Rational DOORS e muito mais. O codebeamer está disponível como uma plataforma SaaS no local ou hospedada na nuvem com opções de licenciamento flexíveis para atender às necessidades de cada equipe. Um teste gratuito de 30 dias é fornecido. Informações sobre preços estão disponíveis mediante solicitação.

**Gerenciamento de serviço Jira (https://www.atlassian.com/br/software/jira)**

Melhor ferramenta de gerenciamento de requisitos + help desk ITSM. O Jira Service Management permite atender aos requisitos do projeto e da equipe em um painel unificado para total transparência operacional.

O Jira Service Management é um software de gerenciamento de requisitos que ajuda as equipes a gerenciar e resolver solicitações de atendimento ao cliente de forma eficiente e colaborativa.

O Jira Service Management é um poderoso software de gerenciamento de serviços de TI que permite que as organizações simplifiquem e automatizem suas operações de service desk. Os principais recursos incluem gerenciamento de incidentes, gerenciamento de problemas, gerenciamento de mudanças e gerenciamento de solicitações de serviço, todos totalmente personalizáveis para atender às necessidades exclusivas de cada organização. Outros recursos incluem um portal de autoatendimento para que os usuários finais enviem solicitações e rastreiem seu status, rastreamento e relatórios de SLA, regras de automação para roteamento e escalonamento de tíquetes e

integração com outras ferramentas e serviços, como Confluence e Slack. O Jira Service Management também oferece recursos avançados de análise e geração de relatórios, permitindo que as organizações acompanhem seu desempenho e identifiquem áreas para melhoria.

O Jira Service Management pode ajudar no gerenciamento de requisitos, permitindo que os usuários capturem, rastreiem e gerenciem os requisitos à medida que evoluem ao longo do processo de desenvolvimento. No Jira Service Management, os usuários podem criar e rastrear requisitos usando campos personalizados, fluxos de trabalho e tipos de problemas específicos para as necessidades de sua organização. Além disso, os usuários podem vincular os requisitos a outros problemas relacionados, como bugs ou solicitações de recursos, para garantir que todo o trabalho necessário seja concluído antes que o requisito seja considerado 'concluído'. O Jira Service Management também oferece relatórios robustos e ferramentas de visualização que podem ajudar as partes interessadas a entender o status e o progresso dos requisitos, bem como identificar possíveis obstáculos ou áreas para melhoria.

O Jira Service Management se integra a outras plataformas da Atlassian, bem como a aplicativos de terceiros, como Slack, Mircosoft, Google Workspace, Zoom, AdobeXD, Invision, Figma, Gliffy, Draw.io, Balsamiq, Lucidchart, Miro, Opsgenie, Jenkins, Dynatrace, GitHub, Zendesk, Trello, Optimizely e centenas de outros por meio de seu mercado de aplicativos. Você também pode criar o seu próprio usando a API deles.

**Doc Sheets (https://asana.com/pt/apps/google-sheets)**

O melhor software intuitivo de gerenciamento de requisitos empresariais. O Doc Sheets permite identificar, organizar e documentar requisitos para análise e comunicação em qualquer ciclo de vida do sistema.

O Doc Sheets é uma plataforma projetada para fornecer gerenciamento de requisitos corporativos intuitivo e acessível para equipes e empresas de todos os tamanhos. O Doc Sheets é rápido e fácil de configurar, além de ser uma solução segura, escalonável e de alto desempenho para gerenciamento de mudanças, gerenciamento de projetos, gerenciamento de casos de teste e recursos de fluxo de trabalho.

Sem qualquer codificação, o Doc Sheets é personalizável para qualquer projeto, quer você o obtenha como uma nuvem (SaaS) ou uma solução local. Permite a criação e rastreamento de requisitos para verificação e validação de sistemas. Ele também acomoda diferentes processos e metodologias para trabalho ágil, scrum e cascata. O Doc Sheets capacita os processos de rastreabilidade e fornece relatórios de rastreabilidade e análise de lacunas quando necessário.

Os principais recursos incluem a capacidade de classificar e organizar requisitos em um repositório robusto; gerar automaticamente documentos de especificações como PDFs, HTML ou DOCX; requisitos de filtro e grupo para relatórios; exportar seus dados para documentos Excel e/ou Word; requisitos de importação de várias fontes externas; versões de gerenciamento automático e lançamentos de produtos; salvar modelos de informações do produto para acesso fácil mais tarde; e simplifique a rastreabilidade com um conjunto de ferramentas abrangente.

Com colaboração em tempo real, vários usuários podem contribuir para a documentação de requisitos simultaneamente. Os relatórios de impacto direto e indireto da mudança podem cobrir suas bases à medida que as coisas mudam, mudam e crescem. Além disso, o controle de acesso e a segurança garantem que apenas as mãos aprovadas estejam no convés para cada projeto ou item.

O Doc Sheets se integra ao JIRA. Também permite importar dados de várias fontes (JSON, Excel etc.).

**ReqTest (https://reqtest.com/)**

Melhor para personalização. Se você está procurando uma ferramenta fácil de implementar, acessar e aprender a usar, o ReQtest deve estar no topo da sua lista. É uma ferramenta totalmente baseada em nuvem que você pode acessar de qualquer lugar, com um modelo de preços amigável e transparente que acomoda intencionalmente equipes pequenas, ágeis e dimensionáveis (bem como grandes organizações) com uma abordagem de "pague pelo que usar".

O ReQtest oferece um conjunto completo de ferramentas úteis de gerenciamento de requisitos, bem como recursos projetados para coleta ágil de requisitos. Isso inclui ferramentas de gerenciamento de teste, rastreamento de bugs e ferramentas visuais de relatórios de bugs, rastreabilidade de requisitos de ponta a ponta, fácil exportação de dados para Excel, recursos de colaboração e quadros de tarefas ágeis.

Esta é uma solução leve, mas eficaz, que funciona bem se você estiver satisfeito com uma ferramenta autônoma (o ReQtest possui sincronização bidirecional com o Jira, mas nenhuma outra integração pré-criada) e não precisa dela para executar as funções de um sistema QMS ou ALM completo.

No geral, é uma das soluções com preços mais acessíveis por aí. Começa em $ 10/usuário/mês, tornando-se uma solução introdutória de baixo risco, especialmente para organizações que estão usando uma ferramenta de gerenciamento de requisitos pela primeira vez.

Com todas essas ferramentas, verificações e validações dos requisitos de software, verificamos que, realmente, os profissionais que irão atuar nessa área tem muitos argumentos para desenvolver requisitos de qualidade e com uma grande precisão, evitando, assim, os erros que podem acontecer.

As ferramentas de gerenciamento de requisitos só vêm para poder dar mais um auxílio, tornando todo o trabalho do ciclo de vida dos requisitos muito mais preciso e eficiente.

# LEITURA COMPLEMENTAR

## ENTENDA O QUE É GERENCIAMENTO DE REQUISITOS

Wilson Souza

Quando você gerencia um projeto, existem várias categorias que são trabalhadas durante o processo e o gerenciamento de requisitos é uma delas. Os requisitos definem os detalhes relativos ao escopo, que é a descrição do que se espera do projeto, ou seja, a expressão refere-se à idealização daquele produto ou serviço que deve ser criado, entregue ou feito.

Coletar os requisitos significa definir e documentar as necessidades das partes interessadas para se chegar aos objetivos estabelecidos para o projeto. São eles que escolhem o caminho que deve ser tomado, as funcionalidades, as especificidades de qualidade e desempenho, os detalhes que precisam estar presentes no produto final.

O gerenciamento de requisitos, por sua vez, refere-se ao conjunto de atividades que possibilita a equipe do projeto a identificar, controlar e rastrear esses requisitos, bem como as suas alterações – é o processo que gerencia as mudanças nos requisitos do projeto. Essas mudanças podem decorrer de erros ou confusão no processo de engenharia de requisitos, design ou, até mesmo, problemas de implementação.

Novos requisitos podem surgir também conforme os stakeholders – público que têm participação, investimento ou interesse no negócio – desenvolvem uma melhor compreensão do produto ou serviço em questão, ou em decorrência de mudanças nas circunstâncias externas, por exemplo, novas leis ou regulamentações que, porventura, venham a ser introduzidas no ambiente do empreendimento.

São considerados incompletos ou inconsistentes, os requisitos que não estão de acordo com as principais necessidades ou não satisfaçam as expectativas dos consumidores daquele determinado produto.

QUAIS SÃO AS PRINCIPAIS PREOCUPAÇÕES DE GERENCIAMENTO DE REQUISITOS?

É preciso preocupar-se em:

- Gerenciar mudanças nos requisitos acordados.
- Gerenciar os relacionamentos entre os requisitos.
- Gerenciar as dependências entre o documento de requisitos e outros documentos produzidos ao longo do processo de desenvolvimento do projeto.

QUAIS SÃO AS PRINCIPAIS CATEGORIAS DE REQUISITOS?

Podem existir inúmeros requisitos dentro de um projeto, desde que toda a informação esteja organizada em uma matriz de rastreabilidade, para que todos os dados possam ser conferidos e bem gerenciados durante a construção do projeto.

Dentro das categorias dos requisitos, os mais comuns são os chamados requisitos técnicos, que especificam de que maneira o produto deve ser feito; os funcionais, que tratam do que o produto deve ser capaz ou não de fazer; e os de qualidade, que se referem à desenvoltura que se espera do produto, depois de pronto.

Fonte: https://blog.acelerato.com/projetos/entenda-o-que-e-gerenciamento-de-requisitos/. Acesso em: 18 abr. 2023.

# RESUMO DO TÓPICO 3

**Neste tópico, você aprendeu:**

- As empresas buscam, por exemplo, profissionais que saibam extrair da grande massa de dados gerada diariamente as informações que sejam relevantes para o funcionamento dos negócios.

- O processo de verificação, que garante que o software atenda às especificações, já o processo de validação irá se concentrar em saber se o software atende às expectativas e requisitos do usuário final. Desenvolvimento de Requisitos, Verificação de Requisitos, Validação de Requisitos, Verificação de Sistema e Validação do sistema são tarefas importantes. Este papel começa com uma definição e explicação desses termos.

- As ferramentas e softwares para ajudar no processo de gerenciamento de requisitos. Essas ferramentas, chamadas coletivamente de ferramentas de engenharia de requisitos, podem facilitar muito a vida dos gerentes de engenharia. Dada a natureza exigente dos locais de trabalho modernos, é essencial que os gerentes de engenharia aprendam essas ferramentas, agregando valor ao seu trabalho.

- As ferramentas de engenharia de requisitos (RE), que estão evoluindo rapidamente. A demanda por flexibilidade e desenvolvimento ágil, colaboração mundial, ecossistemas avançados de software e sistemas está mudando a forma como gerenciamos os requisitos.

# AUTOATIVIDADE

1 A importância de ter um processo de ER adequado em vigor, que produza requisitos suficientemente bons, pode ser considerada crucial para o desenvolvimento bem-sucedido de produtos, seja em um esforço de desenvolvimento sob medida ou orientado para o mercado. No entanto, há indicações claras de que a engenharia de requisitos está faltando na indústria, uma vez que as inadequações nos requisitos são um dos principais determinantes do fracasso do projeto. Muitos dos desafios são relevantes tanto para RE quanto para MDRE sob medida. Por exemplo, problemas (qualidade inadequada) nos requisitos filtram para projetar e implantar. Sobre os resultados publicados por Davis (1993), assinale a alternativa CORRETA:

a) ( ) Poderia ser até 200 vezes mais caro detectar e reparar defeitos durante a fase de manutenção de um sistema em comparação com a engenharia de requisitos, e várias outras fontes indicam que requisitos inadequados são a fonte principal para o fracasso do projeto. Isso é ainda agravado no caso do MDRE como seleção inicial de requisitos é crucial, em particular porque grandes volumes de requisitos de várias fontes correm o risco sobrecarregar as empresas.
b) ( ) A maioria dos requisitos já são declarados de uma forma ou de outra quando atingem o engenheiro de requisitos, tornando-os mais baratos de elaborar.
c) ( ) Muitas organizações se deparam com uma mistura de tendências voltadas para o mercado e desenvolvimento de produtos sob medida.
d) ( ) Essa situação torna necessário adotar RE sob medida para, por exemplo, projetos de adaptação, ao mesmo tempo em que possui um processo MDRE que lida com a engenharia de requisitos contínuos e que rege o desenvolvimento de produtos para o mercado.

2 O processo de verificação garante que o software atenda às especificações. A validação se concentra em saber se o software atende às expectativas e requisitos do usuário final. Desenvolvimento de Requisitos, Verificação de Requisitos, Validação de Requisitos, Verificação de Sistema e Validação do sistema são tarefas importantes. Este papel começa com uma definição e explicação desses termos. Então, dá duas dúzias de exemplos de falhas de sistema famosas e sugere os erros que podem ter sido cometidos. Cada requisito deve ser verificado e validado para garantir que seja o requisito correto. Isso garante que os requisitos atendam ao objetivo geral do sistema e a todas as necessidades das partes interessadas. A verificação e a validação devem ser feitas continuamente ao longo do desenvolvimento de requisitos em todos os níveis e, como parte das atividades de linha de base, devem ser revisadas durante as Revisões de Requisitos do Sistema (SRR) . Os termos para verificação e validação são muitas vezes confusos. Sobre a definição para entender verificação e validação de requisitos, assinale a alternativa INCORRETA:

a) ( ) Validarum sistema.
b) ( ) Verificação de requisitos
c) ( ) Requisitos de validação
d) ( ) Requisitos de complexidade.

3 A validação é uma etapa crítica para encontrar requisitos ausentes e garantir que os requisitos tenham uma variedade de características importantes. Sobre o que a validação de software aborda, classifique V para as sentenças verdadeiras e F para as falsas:

( ) Argumento lógico.
( ) Implementar e desmembrar.
( ) Facilmente vinculado aos requisitos do sistema, como projetos, códigos e testes.

Assinale a alternativa que apresenta a sequência CORRETA:
a) ( ) V – F – F.
b) ( ) V – F – V.
c) ( ) F – V – F.
d) ( ) F – F – V.

4 Imagine você inserindo uma fórmula em uma planilha. Depois de inserir algumas linhas de dados, você pode verificar a fórmula e certificar-se de que está funcionando. O processo de verificação é o mesmo, pois permite que você faça uma verificação rápida antes de se aprofundar no processo de desenvolvimento do produto. Cada requisito deve ser verificado por argumento lógico, inspeção, modelagem, simulação, análise, revisão especializada, teste ou demonstração. Baseado no exposto, indique sobre o que se trata cada verificação de requisito (argumento lógico, inspeção, modelagem, simulação, análise, revisão especializada, teste ou demonstração).

5 O mercado de TI ainda tem muito o que amadurecer no país, mas as empresas já estão começando a perceber a necessidade de analisar os processos e gerenciar necessidades antes de começar um projeto. Afinal, segundo o livro *Software Testing*, de Ron Patton (2001), 60% dos bugs são gerados por problemas de especificação. Além disso, apenas 6% dos projetos de software de grande porte são concluídos com sucesso no mundo todo. Enquanto isso, 52% dos projetos terminam com débito e 42% fracassam. Todo esse prejuízo e problemas durante o desenvolvimento poderiam ser prevenidos se fossem realizadas análises de requisitos e do projeto antes da sua realização, além de documentações para prevenir falhas em projetos futuros. A tendência é que este profissional se torne cada vez mais necessário para evitar prejuízos em projetos na área de TI. Quais seriam as medidas que esse profissional poderia tomar para mudar este cenário que encontramos?

# REFERÊNCIAS


ABREU, F. P.; FARIAS, P. P. M.; ALBUQUERQUE, A. B. A Process to Monitor the Software Acquisition Based on Verification and Validation. *In*: Conference on Research and Practical Issues of Enterprise Information System, **Proceedings**... [*S.l.*], p. xx-xx, Oct. 2009. Disponível em: http://site.ebrapem.org.br/2009/pdf/02-Process.pdf. Acesso em: 19 abr. 2023.

BAHILL, A. T.; HENDERSON, S. Requirements Development, Verification, and Validation exhibited in famous failures. **Systems Engineering**, v. 8, n. 1, p. 1-14, 2005.

BARRETO, A. O. S. **Apoio à Verificação de Software em Ambientes de Desenvolvimento de Softwares Orientados à Organização**. 2006. 172 f. Dissertação (Mestrado em ciências em engenharia de sistemas e computação) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006. Disponível em: https://www.cos.ufrj.br/index.php/pt-BR/publicacoes-pesquisa/details/15/1908. Acesso em: 19 abr. 2023.

BASILI, V.; GREEN, S. Software Process Evolution at the SEL. **IEEE Software**, v. 11, n. 4, p. 58-66, 1994.

PATTON, R. **Software Testing**. Indianapolis: Sams Publishing, 2001.

BLACKBURN, M. R.; BUSSER, R.; NAUMAN, A. **Removing Requirement Defects and Automating Test**. Software Productivity Consortium NFP, 2001. Disponível em: https://www.t-vec.com/download/papers/Removing_Defects_Automating_Test.pdf. Acesso em: 19 abr. 2023.

CALVO, M. V., J. A. *et al*. Experiences in the Application of Software Process Improvement in SMEs. **Software Quality Journal**, v. 10, n. 3, p. 261-273, 2002.

COELHO, J. **Gerenciamento de Requisitos**. 2014. Disponível em: https://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx. Acesso em: 20 mar. 2023.

COVENTRY, T. Gerenciamento de requisitos – planejando para o sucesso: técnicas para acertar no planejamento de requisitos. *In*: PMI® Global Congress 2015—EMEA, **Proceedings**... Londres: Project Management Institute, 2015.

DAVIS, A. M. **Software Requirements**: Objects, Functions, and States. Englewood Cliffs: Prentice Hall, 1993.

DORFMAN, M.; THAYER, R. **Software Engineering**. Los Alamitos: IEEE Computer Society Press, 1997. p. 80.

ENGHOLM, H. **Engenharia de Software na Prática**. São Paulo: Novatec, 2010.

FABBRINI, F. *et al*. **An automatic quality evaluation for natural language requirements**. 2001. Disponível em: https://www.researchgate.net/publication/244206643_An_Automatic_Quality_Evaluation_for_Natural_Language_Requirements. Acesso em: 19 abr. 2023.

GLINZ, M. **A Lightweight Approach to Consistency of Scenarios and Class Models**. 2020. Disponível em: https://ieeexplore.ieee.org/document/855584. Acesso em: 19 abr. 2023.

IEEE. **IEEE Recommended Practice for Software Requirements Specifications**. IEEE Std 830-1993, 1993. Disponível em: https://personal.utdallas.edu/~chung/RE/IEEE830-1993.pdf. Acesso em: 19 abr. 2023.

KANNENBERG, A.; SAIEDIAN, H. Why software requirements traceability remains a challenge. **CrossTalk: The Journal of Defense Software Engineering**, v. 22, n. 4, p. 10-14, jul./aug. 2009.

MACHADO, F. N. R. **Análise e gestão de requisitos: onde nascem os sistemas**. São Paulo: Érica, 2011.

MATIELLO-FRANCISCO, M. F. *et al*. Verificação e validação na terceirização de software embarcado em aplicações espaciais. *In*: V SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, n. 5, 2006, Vila Velha - ES. **Anais** [...]. São José dos Campos: SBQS, 2006.

MOREIRA, A. R.; ARAÚJO, J.; BRITO, I. **Crosscutting quality attributes for requirements engineering**. 2002. Disponível em: https://www.researchgate.net/publication/221390418_Crosscutting_quality_attributes_for_requirements_engineering. Acesso em: 19 abr. 2023.

PFLEEGER, S. L. **Engenharia de software**: teoria e prática. 2. ed. São Paulo: Pearson Education do Brasil, 2004.

PRESSMAN, R. S. **Engenharia de software**. 6 ed. Rio de Janeiro: Mcgraw-Hill Interamericana, 2006.

RAMESH, B.; JARKE, M. Towards reference models for requirements traceability. **IEEE Transactions on Software Engineering**, v. 27, n. 1, p. 58-93, Jan. 2001.
ROCHA, F. *et al*. Uma proposta de modelo para avaliar a qualidade da tradução de requisitos para casos de uso. *In*: SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 3., 2006, Curitiba. **Anais**... Curitiba: SBES, 2006.

ROSENBERG, L. *et al*. Requirements, testing, and metrics. *In*: PACIFIC NORTHWEST

SOFTWARE QUALITY CONFERENCE, 15th, 1998, Utah. **Proceedings**... Utah, 1998.

RYSER, J.; GLINZ, M. **A practical approach to validating and testing software systems using scenarios**. 1999. Disponível em: https://www.semanticscholar.org/paper/A-Practical-Approach-to-Validating-and-Testing-Ryser-Glinz/108ec8667b26cd6a6b94849f648585ad82447f33. Acesso em: 19 abr. 2023.

SOMMERVILLE, I.; SAWYER, P. **Requirements Engineering**: A Good Practice Guide. Chichester: John Wiley & Sons, 1997.

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

TRAVASSOS, G.; GUROV, D.; AMARAL, E. **Introdução à Engenharia de Software Experimental**. Rio de Janeiro: COPPE/UFRJ, 2002.

ZAHRAN, S. **Software process improvement: practical guidelines for business success**. Reading, MA: Addison-Wesley, 1998.

WIEGERS, K. E. **Software requirements: practical techniques for gathering and managing requirements throughout the product development cycle**. Redmond: Microsoft Press, 2003.