

# **Engenharia de Requisitos e Processos de Software**



**Cruzeiro do Sul Virtual**  
Educação a distância





## Engenharia de Requisitos e Processos de Software

**Responsável pelo Conteúdo:**

Prof. Me. Erwin Alexander Uhlmann

**Revisão Textual:**

Prof.<sup>a</sup> Dra. Selma Aparecida Cesarin



# UNIDADE

## Engenharia de Requisitos e Processos de *Software*



- Introdução;
- Estudos dos Tipos de Requisitos;
- Requisitos Funcionais;
- A Natureza dos Requisitos Funcionais;
- Completeza e Consistência: Os Pilares da Especificação – Exemplificando a Aplicação dos Conceitos;
- Desafios e Soluções na Especificação de Requisitos;
- Requisitos Não Funcionais;
- Técnicas de Elicitação de Requisitos;
- Verificação, Análise e Gerenciamento dos Requisitos;
- Processo de *Software*;
- Modelo Cascata;
- Modelo de Desenvolvimento Cascata;
- Desenvolvimento Evolucionário;
- Processo de Desenvolvimento Iterativo;
- Desenvolvimento Incremental;
- Modelo Espiral.



### OBJETIVOS DE APRENDIZADO

- Compreender os principais requisitos necessários com a respectiva classificação de seus elementos;
- Elaborar especificações detalhadas de Projetos de Sistemas que possam ser implementadas em máquinas e plataformas computacionais orientadas a objetos.



# Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



## Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

# Introdução

Entender os princípios que dão base às coisas é crucial para compreender seus propósitos e metas.

Qual é a função de um *software*?

*Software* consiste em um conjunto de Códigos que se interligam logicamente para gerar um resultado desejado, baseando-se na configuração e nos dados inseridos. Em linhas gerais, o papel principal de um *software* é gerar saídas lógicas que otimizem a solução de problemas diários e viabilizem a simulação de diversos cenários. Para desenvolver um *software* de maneira eficaz, ou seja, que seja estável, durável e testável, é imprescindível a definição precisa dos requisitos.

Mas o que caracteriza requisitos precisos?

No caso de um *software* estável, é essencial que não haja requisitos contraditórios. Para garantir durabilidade, deve-se cobrir integralmente as demandas, e para que seja testável, deve-se assegurar a verificação do cumprimento das exigências.

Assim, podemos estabelecer um padrão: RAPITO.

- Requisito;
- Análise;
- Projeto;
- Implantação;
- Teste;
- Operação.

# Estudos dos Tipos de Requisitos

## Requisitos Funcionais e Não Funcionais

A Engenharia de Requisitos (ER) constitui um pilar fundamental no desenvolvimento de *software*, diferenciando-se claramente de conceitos como Entidade Relacional em Bancos de Dados. Ao iniciar o processo de solicitação de um *software*, é essencial que a abstração guie a formulação dos requisitos. Essa abstração permite que surjam múltiplas soluções e perspectivas variadas, enriquecendo o processo criativo e técnico na busca por soluções eficazes. Contudo, a fase subsequente, a da proposta, exige um movimento contrário em direção à especificidade. A especificação precisa ser detalhada, clara e objetiva, permitindo uma análise técnica rigorosa. Essa definição precisa permite que as propostas sejam comparáveis entre si, facilitando a identificação da solução mais adequada por meio da avaliação de padrões e consistência.

Nesse contexto, os Requisitos de Usuário desempenham papel crucial. Eles são expressos em uma Linguagem compreensível para os *stakeholders*, utilizando a Linguagem natural, diagramas e outras ferramentas visuais ou escritas, para esclarecer o que se espera do Sistema de maneira abstrata. Essa etapa é crucial para garantir que os desenvolvedores compreendam plenamente os objetivos e as necessidades dos usuários finais.

A transformação desses requisitos abstratos em Requisitos do Sistema, ou especificações funcionais, representa um dos desafios centrais da Engenharia de Requisitos. Essas especificações detalham funções, serviços e restrições operacionais do Sistema, servindo como o projeto para a construção efetiva do *software*. Para realizar essa transição de forma eficaz, é imprescindível a elaboração de um documento comprehensivo que não apenas traduza os Requisitos de Usuário em termos técnicos, mas também detalhe o caminho a ser seguido para satisfazê-los.

Esse processo minucioso de especificação é vital para o desenvolvimento de *software*, pois assegura que todas as partes interessadas – desenvolvedores, clientes e usuários finais – estejam alinhadas com o que será entregue. Ao estabelecer um diálogo claro e preciso desde o início, evitam-se mal-entendidos e ajustes custosos durante as fases de desenvolvimento e implementação. Portanto, a habilidade de abstrair inicialmente, seguida de uma especificação detalhada e objetiva, não é apenas uma boa prática da Engenharia de Requisitos, mas uma necessidade para o sucesso do Projeto de *software*.



O Engenheiro de Requisitos deve converter cada etapa da seguinte maneira:

1. Para alcançar simplicidade, o que é necessário? Implementar uma interface com opções limitadas e realizar a programação de forma concisa;
2. Para garantir funcionalidade, o que é essencial? Criar uma interface cujo *design* e *layout* enfatizem as funcionalidades em vez da estética;
3. Para que os botões sejam intuitivos e demonstrem sua função, o que deve ser feito? Analisar a função primária do botão, evitar incorporar funcionalidades secundárias que desviam do propósito original e adotar um *design* simples;
4. Para modificar dados sem conhecimento em programação, o que é necessário? Elaborar formulários com títulos e questões claras que indiquem sua função de alteração.

## Requisitos Funcionais

Os Requisitos Funcionais são os pilares que orientam a estruturação, o desenvolvimento e a implementação de sistemas, definindo claramente o que o *software* deve fazer, como deve interagir com os usuários e quais resultados são esperados de suas funções. Diferentemente dos requisitos do usuário, que se concentram nas necessidades e nos desejos abstratos do usuário final, os Requisitos Funcionais detalham as especificações técnicas necessárias para transformar essas necessidades em funcionalidades concretas dentro do *software*.

# A Natureza dos Requisitos Funcionais

Os Requisitos Funcionais descrevem as operações específicas, interfaces, comportamentos e outras funcionalidades que um Sistema deve ter para atender às necessidades dos seus usuários. Eles servem como um guia detalhado para desenvolvedores e Engenheiros de Sistemas, especificando como os dados de entrada serão processados pelo Sistema para produzir os dados de saída desejados. Esses requisitos abrangem desde a descrição de processos simples, como o cadastro de usuários, até funções complexas, como a geração de relatórios analíticos ou a realização de transações seguras *on-line*.

## Completeza e Consistência: Os Pilares da Especificação – Exemplificando a Aplicação dos Conceitos

Consideremos o exemplo proposto de um Sistema de Cadastro para funcionários acessarem arquivos do escritório. Os Requisitos Funcionais detalhados, desde o objetivo do software até as especificações de segurança e acessibilidade, são cruciais para o sucesso do Sistema. Cada requisito, como a necessidade de campos de texto para nome e *e-mail* (RF01), seleção de cargo e supervisor (RF02), níveis de acesso definidos (RF03) e compatibilidade com dispositivos móveis (RF04) forma a espinha dorsal do que será o Sistema final.

No entanto, a completeza e a consistência devem sermeticulosamente verificadas. Por exemplo, a indicação do tipo e do tamanho dos campos de texto (RF01) deve ser clara para evitar ambiguidades. Da mesma forma, a descrição de como os dados de cargo e supervisores são gerenciados e atualizados (RF02) precisa ser suficientemente detalhada para evitar inconsistências no uso do sistema.

## Desafios e Soluções na Especificação de Requisitos

Um dos maiores desafios na elaboração de Requisitos Funcionais é garantir que eles sejam ao mesmo tempo completos e consistentes. Inconsistências, como a contradição entre a restrição de acesso por IP e a permissão de dispositivos móveis (RF04), podem levar a problemas significativos na fase de implementação do Sistema. Portanto, é essencial que haja revisão e validação cuidadosas de todos os requisitos para resolver tais contradições antes que o desenvolvimento comece.

A solução para muitos desses desafios reside na comunicação eficaz entre as partes interessadas e a Equipe de desenvolvimento, bem como na utilização de ferramentas e

metodologias de Engenharia de Requisitos. *Workshops*, prototipagem, modelagem e análise de requisitos são técnicas valiosas que podem ajudar a esclarecer, a detalhar e a validar os Requisitos Funcionais.



Os Requisitos Funcionais são fundamentais para o desenvolvimento de Sistemas de Informação eficazes e eficientes. Eles não apenas definem o que o *software* deve fazer, mas também orientam como ele deve ser feito. A elaboração de um documento de requisitos que seja ao mesmo tempo completo e consistente é um passo crucial no processo de desenvolvimento de *software*, garantindo que o produto atenda às necessidades dos usuários e funcione harmoniosamente dentro do seu ecossistema tecnológico e operacional. Ao superar os desafios de especificação de requisitos por meio da comunicação eficaz, revisão cuidadosa e utilização de metodologias apropriadas, é possível alcançar um Sistema robusto, funcional e adaptável às necessidades em constante evolução dos usuários.

Veja os Requisitos Funcionais aplicados na Figura 1, a seguir:

Nome:	CPF:	Data Nascimento:
<input type="text"/>	<input type="text"/>	<input type="text"/> dd/mm/aaaa <input type="button" value="..."/>
e-mail:	Senha:	
<input type="text"/> @crefsp.gov.br	<input type="text"/>	
Departamento ou Seccional:	Cargo:	
<input type="text"/>	<input type="text"/>	
Local Trabalho:	<input type="text"/>	
	<input type="button" value="Gravar"/>	

Figura 1 – Parte dos Requisitos Aplicados. Campo Nome (RF01) sem o descritivo de tamanho (falta de Completeza) que está com o mesmo tamanho do campo e-mail. Para o RF02, o campo Local de Trabalho não tem descrito de onde vem os dados

Fonte: Reprodução

#ParaTodosVerem: a Figura tem oito campos, com as descrições Nome, e-mail, Departamento ou Seccional, Local Trabalho, CPF, Senha, Cargo e Data Nascimento. Os campos Departamento ou Seccional, Local Trabalho e Cargo são do tipo select e o quadro termina com um botão vermelho escrito Gravar. Fim da descrição.

## Requisitos Não Funcionais

Os Requisitos Não Funcionais constituem uma parte crucial do desenvolvimento de Sistemas, focando em aspectos que não estão diretamente ligados às funções específicas que o *software* desempenhará, mas que são essenciais para garantir a eficácia e a adequação do Sistema às necessidades do Projeto. Contrariamente aos requisitos funcionais, que delineiam claramente as operações e as funcionalidades que o sistema é capaz de executar, os requisitos não funcionais detalham as condições e as capacidades necessárias para que o Sistema possa efetivamente cumprir essas funções.

Eles abrangem uma várias especificações, incluindo, mas não se limitando a, escolhas de Linguagem de Programação, como Java ou PHP, seleção de Plataformas, que podem variar desde Sistemas Operacionais locais como *Windows*, *Linux* ou *Unix* até ambientes *web* hospedados em *Apache* ou *IIS*, além dos requisitos de *hardware* necessários para suportar o Sistema.

Esses requisitos são categorizados em três grandes grupos: Requisitos de Produto, que se referem às especificações técnicas e de desempenho do *software*; Requisitos Organizacionais, que dizem respeito aos padrões e às políticas da Organização desenvolvedora; e Requisitos Externos, que incluem regulamentações legais e de interoperabilidade com outros Sistemas.

Cada um desses aspectos não funcionais desempenha papel vital na estruturação e no desenvolvimento de um Sistema robusto, garantindo não só que o *software* funcione conforme o esperado em termos de operações, mas também que esteja em conformidade com as exigências de infraestrutura, Legislação e padrões de qualidade. Dessa forma, os Requisitos Não Funcionais são indispensáveis para a entrega de um Sistema que não apenas atenda às necessidades funcionais dos usuários, mas que também seja sustentável, seguro e eficiente do ponto de vista do ambiente em que será implementado, como mostra a Figura 2, a seguir.

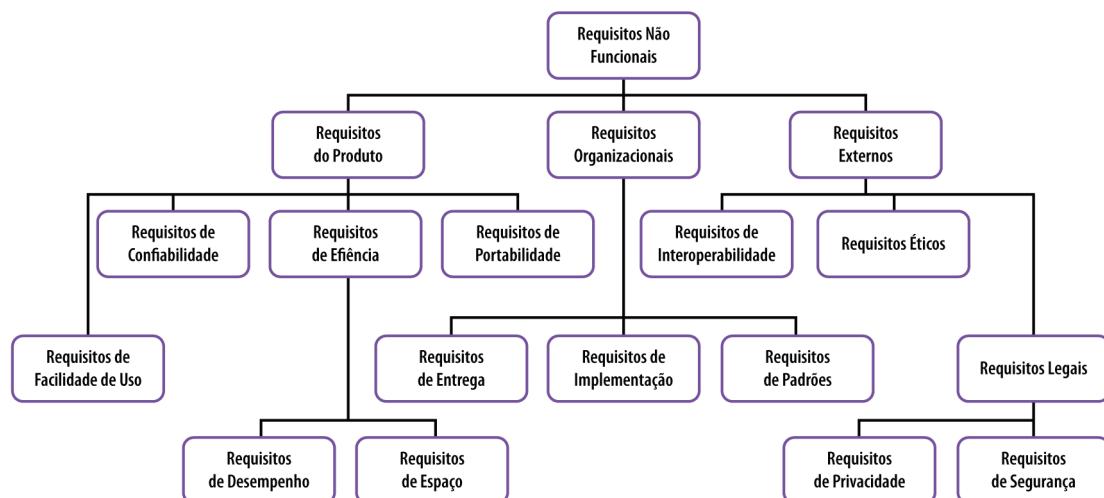


Figura 2 – Quadro de tipos de Requisitos

Fonte: Adaptada de SOMMERVILLE, 2007

**#ParaTodosVerem:** a Figura apresenta a hierarquia de tipos de Requisitos em quadros ligados por linhas na ordem: Requisitos Não Funcionais. Abaixo, ligados a ele, estão os Requisitos do Produto, Requisitos Organizacionais e Requisitos Externos, que levam por nível a: Requisitos de Confiabilidade, de Eficiência, Portabilidade, Interoperabilidade, Éticos, Facilidade de Uso, Entrega, Implementação, Padrões, Legais, Desempenho, Espaço, Privacidade e Segurança.

Para cumprir com o RF01 (RF01 – Os dados Nome e *E-mail* devem ser armazenados em campos do tipo texto no cadastro):

**RNF01:** a implementação deve ocorrer em uma plataforma *WEB*, utilizando PHP; (RNF de Produto):

- Justificativa e análise para a escolha de *WEB* e *PHP*;
- Localização dos servidores de hospedagem;
- Definição do Sistema Operacional: *Windows*, *Linux* ou *Unix*?;
- Especificação do tipo de servidor;
- Estratégias para garantir alta segurança e consistência.

Para o RF02 (RF02 – O cadastro deve incluir campos do tipo *select* para os dados de Cargo, permitindo selecionar apenas supervisores, além de indicar outros funcionários).

**RNF02:** o servidor será determinado pela disponibilidade do Departamento de TI, seguindo a política XPTO da organização (RNF Organizacional).

**RNF03:** novos usuários só poderão ganhar acesso mediante autorização de um supervisor, estabelecendo dependência de liberação que assegura o controle de acesso ao Sistema por meio da codependência.

Para avaliar métricas de eficiência, considera-se o seguinte cenário: uma Rede de Servidor 10/100 conectada a 50 computadores.

Caso todos os computadores solicitem acesso simultâneo ao *software*, a capacidade de transferência de dados é de 100Mbps, ou seja, 100Mbps dividido por 50 computadores. Considerando que cada computador pode requerer até 2Mbps e cada caractere consome até 8 bits, temos uma capacidade de 250.000 caracteres por segundo. O desempenho pode ser afetado pela necessidade de carregar imagens e arquivos adicionais, podendo a eficiência ser melhorada pela redução desses elementos.

Assim como os Requisitos não Funcionais foram especificados, é essencial definir os Requisitos de Produto, Organizacionais e Externos, entre outros.

Por exemplo: vamos imaginar um Sistema de Gestão de estoque para um escritório, com itens variados, tais como:

- **RP01:** o Sistema deve permitir o cadastro de Unidades de medida, identificadas por código e descrição (Figura 3);
- **RP02:** o Sistema deve possibilitar o cadastramento, a alteração e a exclusão de itens (Figura 3).

**Código da Unidade de Medida:**  
exemplo: APA, BLC, CXA, UNI ...

**Descrição da Unidade de Medida:**  
exemplo: Aparelho, Bloco, Caixa, Unidade ...

**Cadastrar**

Figura 3 – Exemplo de RP02

Fonte: Reprodução

#ParaTodosVerem: uma Figura de exemplo, com três botões: Consultar, Alterar e Excluir, de cor cinza, e dois campos, o primeiro com o descriptivo “Código da Unidade de Medida” e o segundo “Descrição da Unidade de Medida”, e dois campos para digitação, finalizado com um botão vermelho Cadastrar”. Fim da descrição.

## Técnicas de Elicitação de Requisitos

A elicitação e a análise de requisitos envolvem a colaboração entre a Equipe de Engenharia e o cliente, seguindo um modelo espiral para a elaboração do documento.

Esse modelo se divide em quatro etapas principais:

1. Obtenção dos Requisitos, em que se identificam as necessidades;
2. Classificação e Organização, etapa em que se ordenam e estruturam os requisitos;
3. Priorização e Negociação, momento de definir a importância de cada requisito e ajustar expectativas; e
4. Documentação, fase de registro formal. Essas etapas são visualmente representadas na Figura 4, a seguir.

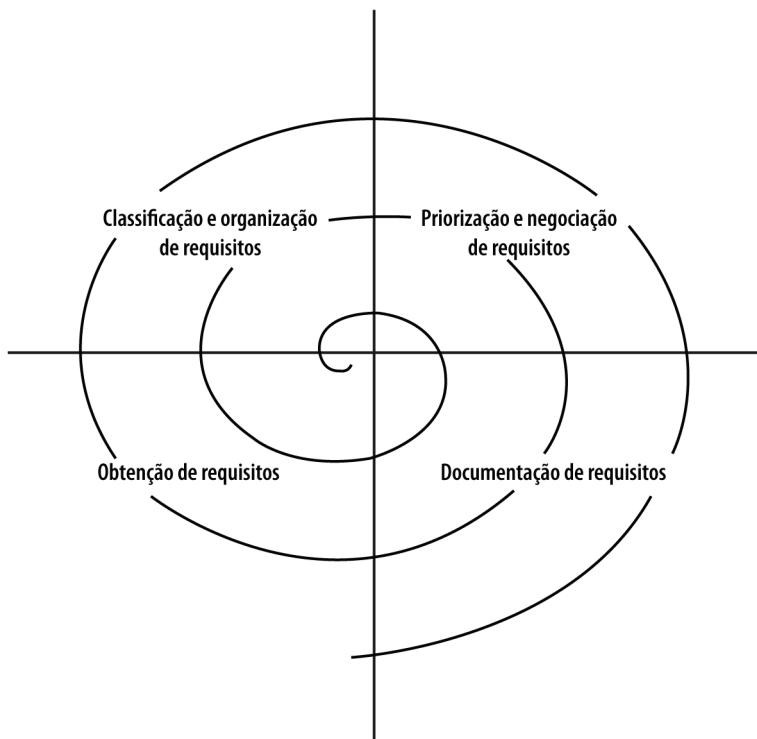


Figura 4 – Processo espiral de Elicitação e Análise de Requisitos

Fonte: Adaptada de SOMMERVILLE, 2007

**#ParaTodosVerem:** gráfico em espiral, dividido em quadrantes, com os itens em cada um deles: Obtenção dos requisitos no primeiro, Documentação de requisitos no segundo, Priorização e negociação de requisitos no terceiro e Classificação e organização de requisitos no quarto. Fim da descrição.

A captação de requisitos é uma etapa crucial na Engenharia de *Software*, envolvendo técnicas como entrevistas diretas, aplicação de questionários tanto quantitativos quanto qualitativos, elaboração de diagramas e análise profunda do negócio em questão. É importante destacar que a responsabilidade de coletar esses requisitos recai sobre o Engenheiro de *Software* e não sobre o cliente. Dado que o cliente geralmente não possui a expertise técnica, quaisquer falhas na captação de requisitos devem ser atribuídas ao profissional de Engenharia de *Software*. Portanto, é essencial que o Engenheiro formule perguntas alinhadas com os conceitos e práticas estudadas até o momento.

Quanto à classificação e à organização dos requisitos, eles devem ser categorizados como relacionados ou complementares e agrupados de maneira lógica e coesa. Uma abordagem eficaz é adotar o modelo *CRUD* (*Create, Read, Update, Delete*), analisando os requisitos de criação, exclusão, leitura e edição em blocos separados. Outra estratégia consiste em dividir os requisitos por módulos, como interface do usuário, regras de negócio e Banco de Dados, garantindo que todos os elementos estejam alinhados em termos de funcionalidade e restrições.

A priorização e a negociação de requisitos surgem como passos seguintes, essenciais para resolver conflitos e redundâncias entre os diferentes *stakeholders*. Esse processo envolve determinar quais requisitos têm precedência e apresentá-los de forma clara para decisão. Por exemplo, em um Sistema Bancário, deve-se ponderar se os requisitos de segurança são mais críticos do que os de edição de dados do usuário. Da mesma forma,

em um Sistema de Registro de pacientes de uma clínica, é necessário avaliar o que tem maior peso: a conformidade com a Lei Geral de Proteção de Dados (LGPD) ou a geração de relatórios.

Após essas etapas, a documentação dos requisitos torna-se o resultado, compreendendo tanto os Requisitos Funcionais quanto os não Funcionais, formalmente descritos para que os desenvolvedores possam prosseguir com o trabalho de maneira consistente e assertiva. A especificação é a fase na qual o documento de requisitos é analisado e traduzido para um formato padrão, como a Linguagem de Modelagem Unificada (UML), o que é crucial para identificar e resolver possíveis conflitos entre os requisitos.

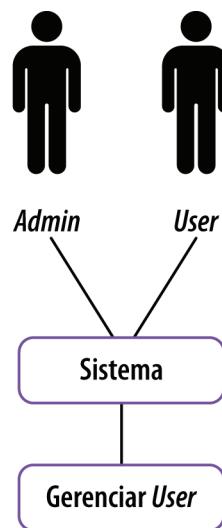


Figura 5 – Especificação com UML com exemplo de conflito entre usuários

Fonte: Adaptada de Freepik

#ParaTodosVerem: imagem de dois bonecos tipo palito descritos como *Admin* e *User*, que se ligam a um balão com a palavra *Sistema* inserida e esse balão se liga a outro balão, com o descriptivo *Gerenciar User*. Fim da descrição.

## Verificação, Análise e Gerenciamento dos Requisitos

O processo de verificação envolve assegurar que os requisitos sejam implementados de maneira eficaz, evitando qualquer falha de especificação. Isso exige uma avaliação criteriosa de cada requisito:

- **Validação:** durante a validação, é crucial examinar se os requisitos são atendidos ou não, identificando possíveis conflitos, redundâncias e falhas, o que requer revisão em diversos aspectos:
  - » **Validade:** é fundamental garantir que todos os usuários, sejam eles Administradores ou Usuários Comuns (conforme ilustrado na Figura 5), tenham a capacidade de realizar o que foi solicitado, acessando os mesmos dados;

- » **Completeza:** precisa-se verificar se todos os requisitos necessários foram considerados para solucionar completamente o problema, assegurando que não haja omissão de detalhes funcionais. Por exemplo, um usuário comum deve ser capaz de visualizar apenas seus próprios dados;
- » **Consistência:** deve-se confirmar que não existam requisitos conflitantes. Por exemplo, enquanto somente o Administrador deve ter a capacidade de gerenciar usuários, todos os usuários precisam poder gerenciar seus próprios dados;
- » **Realismo:** é imprescindível avaliar a viabilidade da execução dos requisitos, considerando prazos, custos e se a infraestrutura e tecnologia existentes são adequadas;
- » **Quantificável, Mensurável e Verificável:** desde a fase de levantamento, os requisitos devem ser claramente Quantificáveis, expressando, por exemplo, o número de funções, clientes, usuários e produtos. Eles também devem ser Mensuráveis, permitindo a avaliação de aspectos como tempo de uso e carga do sistema, e Verificáveis, facilitando a compreensão por parte dos clientes ou usuários de que suas solicitações foram cumpridas;
- » **Casos de Teste:** esta etapa não se refere ao teste do *software* já em funcionamento, mas à criação de cenários específicos, como dois tipos de usuários realizando as mesmas funções ou um mesmo tipo de usuário executando ações contrárias e violando regras de negócio. A elaboração de um Mapa Mental pode ser útil nessa fase.

## Processo de Software

O Processo de Desenvolvimento de *Software* engloba um conjunto de etapas responsáveis pela criação do *software*. Essa sequência de ações é denominada assim devido à integração de várias atividades humanas, técnicas e produtivas, que incluem, respectivamente, a inovação, a elaboração de documentação e a codificação, e a interdependência entre múltiplas atividades de produção simultâneas.

Dentro desse processo, destacam-se as seguintes fases:

- **Especificação:** delimita as funcionalidades, operações e limitações do *software*;
- **Projeto e Implementação:** desenvolvimento conforme as especificações estabelecidas;
- **Validação:** confirmação junto ao cliente de que os requisitos foram atendidos;
- **Evolução:** adaptação às demandas variáveis do cliente.

O Processo de Desenvolvimento de *Software* é primordialmente adotado para fornecer uma documentação que facilite a continuidade do trabalho por outros desenvolvedores que não participaram do Projeto original, sem comprometer a qualidade do produto.

É vital reconhecer que o Processo de Desenvolvimento de *Software* transcende a própria noção de “Processo de *Software*”. A produção do *software* é tão ou mais significativa que o *design* do sistema em si. Para simplificar, pode-se comparar que, assim como um

carro representa o processo de *software*, a existência desse carro dentro de uma “fábrica de *software*” depende do projeto de sua linha de produção. Esse planejamento é tão ou mais complexo do que o *design* do carro, ou do *software*, propriamente dito.



Considere este cenário: será que para cada *software* desenvolvido por nossa Empresa, vamos criar um sistema de *login* e senha do zero? Como, então, poderíamos organizar um repositório, ou uma pasta específica, em que o Sistema de Autenticação esteja pronto para ser integrado a outros sistemas? Qual deveria ser o aspecto do *design*? Ele precisa ser flexível. E quanto à estrutura do Banco de Dados? É essencial utilizar nomes padronizados. Além disso, os identificadores das sessões de *login* e senha devem ser consistentes em todos os sistemas. Assim, nosso processo de desenvolvimento de *software* deve considerar não apenas o projeto em si, mas também a estratégia para facilitar a produção de sistemas, de forma escalável.

Para projetar um sistema de forma adequada, é importante fragmentar em partes menores e compreensíveis, computáveis, as necessidades expressas pelo cliente, traduzidas em Requisitos que permitirão, passo a passo, a validação do Sistema, sem, é claro, desperdiçar recursos como tempo, trabalho ou dinheiro, isto é, não é sair programando para mostrar funcionando, é preciso confirmar passo a passo com o cliente se o entendido está correto, se o Projeto está de acordo, se as funções estão respeitando o Projeto.

Os modelos de Processo de *Software* são abstrações das atividades que auxiliam a compreensão das atividades de produção, entre os modelos podem ser citados os apresentados a seguir.

## Modelo Cascata

O Modelo de Desenvolvimento Cascata, estabelecido na década de 1970, delineia um processo sequencial para o ciclo de vida do *software*. Esse modelo é distintivo por sua abordagem linear, em que cada fase do desenvolvimento é construída sobre a anterior e serve de alicerce para a subsequente, garantindo que cada passo seja tanto um ponto de partida quanto um mecanismo de validação para o próximo. Idealmente aplicado em cenários nos quais as expectativas do cliente e do desenvolvedor são precisas e bem definidas, o Modelo Cascata visa a alcançar resultados que atendam rigorosamente aos prazos, padrões de qualidade e estimativas de custo previamente estabelecidos.

## Modelo de Desenvolvimento Cascata

- **Origem:** Anos 1970;

### Características:

- Definição sequencial de atividades;

- Cada etapa serve de base e é validada pela próxima.

#### **Utilização:**

- Recomendado quando cliente e desenvolvedor têm clareza das expectativas;
- Espera-se obter resultados dentro de prazos, qualidade e custos planejados.

#### **Etapas do Modelo Cascata:**

- Análise e definição de Requisitos:
  - » Documentação realizada com o cliente;
  - » Serve de base para todas as atividades do modelo.

#### **Projeto do Sistema:**

- Definição abstrata da Arquitetura do Sistema;
- Inclui *hardware*, *software* e suas relações e dependências.

#### **Implementação e Teste de Unidade:**

- Teste de cada unidade em partes;
- Teste da unidade completa conforme especificações de requisitos.

#### **Integração e Teste de Sistema:**

- Verificação das relações e dependências após o Teste de Unidade;
- Liberação do *software* para o cliente.

#### **Operação e Manutenção:**

- Fase mais longa do ciclo de vida do *software*;
- Detecção de erros não identificados na produção.

Aprimoramento, ampliação e implementação de novos módulos.

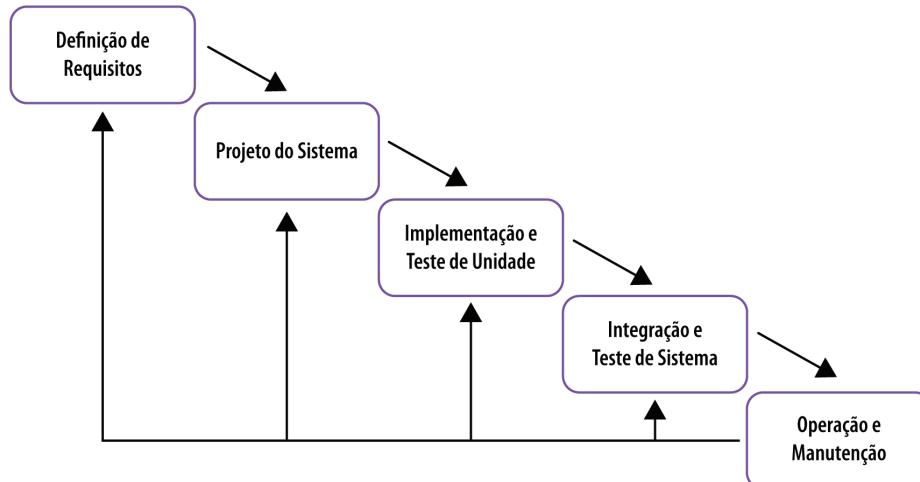


Figura 6 – Processo de Software em modelo cascata

Fonte: Adaptada de SOMMERVILLE, 2007

#ParaTodosVerem: a Figura representa 5 caixas descendentes como em uma escada ou cascata com setas interligando-as na diagonal da primeira para segunda e desta

para a terceira e assim até a quinta, inscritas com os nomes das etapas. Na primeira, Definição de Requisitos; na segunda, Projeto do sistema; na terceira, Implementação e Teste de Unidade; na quarta, Integração e Teste de Sistema e na quinta e última, Operação e Manutenção. Da última, uma seta sai de forma ortogonal e se liga à quarta, à terceira, à segunda e à primeira. Fim da descrição.



De maneira resumida e abrangente, as versões Alfa representam o estágio inicial de liberação. Esse estágio é dividido em três etapas principais:

- **Alfa 1:** nesta fase, o *software*, ainda em desenvolvimento, é testado internamente pela Equipe, apresentando numerosas falhas potenciais ou componentes ausentes;
- **Alfa 2:** após ajustes internos, a versão é disponibilizada ao cliente para testes em um ambiente controlado. Se o *software* é inédito, funciona em conjunto com procedimentos existentes. Se for para substituição, deve ser operado em paralelo ao Sistema antigo, com serviços duplicados para a identificação de erros;
- **Alfa 3:** depois de realizar correções baseadas nos *feedbacks* internos e externos, a versão é liberada para aprimoramentos em usabilidade, acessibilidade, estética e para incorporar novas necessidades identificadas após o teste com o cliente;
- **Beta 1:** frequentemente considerada uma continuação do Alfa 3, essa fase é marcada pela liberação para uso prático;
- **Beta 2:** implementação de correções baseadas nas observações do cliente após o uso prático.
- **Beta 3:** versão definitiva. Incorpora ajustes pós-uso prático, novas funcionalidades identificadas durante o Alfa 2, e refinamentos em usabilidade e acessibilidade, após testes extensivos em condições reais, sem operações paralelas.

De forma geral, também é interessante definir prazos para a mudança de versão. Esses prazos são compatíveis com o porte e a complexidade do *software*. De forma simples, cerca de 10% do tempo de produção, ou seja, se o *software* levou um ano para chegar no Alfa 1, 30 a 40 dias são necessários para a realização dos testes.

#### Entendendo a Estrutura de Versões de um *Software*:

- **Lançamento:** quando um *software* é inicialmente disponibilizado após sua fase de Testes Beta, ele é designado como versão 1, que marca um importante marco de estabilidade e prontidão para o uso geral;
- **Exemplo:** após a Beta 3, o *software* é liberado como versão 1.

#### Atualizações e correções:

- **Melhorias menores:** a introdução de correções ou aprimoramentos em módulos específicos resulta em uma mudança no segundo número da versão, levando a versões como 1.1;
- **Exemplo:** correções iniciais ou melhorias resultam na versão 1.1;
- **Correções subsequentes:** se um módulo já atualizado (como o 1.1) passar por novas alterações, isso é refletido no terceiro número, levando a versões como 1.1.1;
- **Exemplo:** novas alterações em um módulo corrigido levam à versão 1.1.1.

### Mudanças maiores:

- **Reescrita completa:** para alcançar uma versão significativamente nova, como passar para a versão 2, o *software* deve passar por uma reescrita completa, indicando mudanças fundamentais;
- **Exemplo:** uma reescrita total transforma o *software* na versão 2;
- **Reescrita de Módulos:** similarmente, para que um Módulo específico avance para uma nova versão maior (como de 1 para 1.2), ele deve ser integralmente reescrito;
- **Exemplo:** a reescrita de um Módulo específico resulta na versão 1.2;
- **Versões complexas:** com o tempo e o desenvolvimento contínuo, um *software* pode alcançar estruturas de versão mais complexas, como 1.2.3.4.5, que detalham não apenas as grandes revisões, mas também ajustes finos, correções menores e adições específicas;
- **Detalhamento:** essas versões complexas servem para documentar de forma precisa cada pequena mudança, podendo indicar desde melhorias maiores até a inclusão de um novo campo ou botão no sistema.

Essa estrutura de versionamento é crucial para desenvolvedores, usuários e Equipes de Suporte Técnico, pois fornece um registro detalhado do progresso do *software* e das mudanças implementadas ao longo do tempo.

## Desenvolvimento Evolucionário

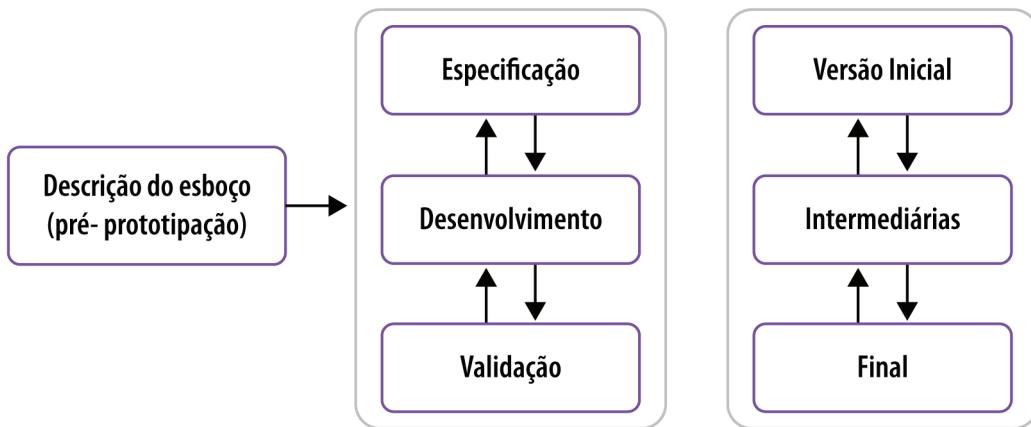
Este processo de produção é caracterizado pela sua natureza quase intangível, em que não é possível estabelecer prazos definidos ou garantir a viabilidade econômica da geração de documentos para cada atualização, frequentemente empregado para satisfazer demandas específicas de clientes que enfrentam dificuldades em expressar suas necessidades ou em entender até que nível o *software* deve ser desenvolvido, tanto por parte do cliente quanto do Engenheiro de Software.

Tais sistemas são projetados para encontrar soluções para desafios que ainda não foram plenamente compreendidos pelo cliente, sem um resultado esperado claro.

Podemos citar como exemplos de aplicação desses sistemas os robôs autônomos capazes de reconhecer voz, gestos e sons, equipamentos cirúrgicos avançados, sistemas de cálculo de variáveis ambientais e análises qualitativas, dentre outros.

Esse projeto pode ser categorizado em dois métodos principais de desenvolvimento:

- **Desenvolvimento Exploratório:** nessa modalidade, à medida que novas funcionalidades são implementadas, demandas adicionais são identificadas e requisitadas pelo cliente;
- **Desenvolvimento Descartável (*Throwaway*):** aqui, o cliente não tem clareza sobre todas as possibilidades que o Sistema pode oferecer ou sobre os requisitos específicos. Um protótipo é, então, desenvolvido com base na interpretação das informações fornecidas pelo cliente, que avalia os resultados para decidir se haverá uma evolução do projeto ou sua descontinuação.

Figura 7 – Modelo *Throwaway*

Fonte: Adaptada de SOMMERVILLE, 2007

#ParaTodosVerem: Figura que apresenta três quadros, um à esquerda, com a inscrição Descrição do esboço (pré-prototipação), um no centro, com outros três quadros internos dispostos verticalmente, com as inscrições Especificação no mais alto, Desenvolvimento no do centro e Validação no mais baixo. No último quadro, à direita, também há 3 quadros internos dispostos verticalmente, com as inscrições Versão Inicial, Intermediárias e Final. Todos os quadros são interligados por setas nos dois sentidos. Fim da descrição.

## Processo de Desenvolvimento Iterativo

Ao longo da criação ou do uso, um *software* pode passar por diversas alterações, motivadas pela implementação de novas Tecnologias ou pela inclusão de requisitos adicionais. Para gerenciar essas mudanças, adota-se o processo de desenvolvimento iterativo, incremental ou faseado.

## Desenvolvimento Incremental

Em situações em que os *softwares* são complexos ou têm componentes do Projeto ainda indefinidos, opta-se pelo desenvolvimento incremental. Essa abordagem analisa o segmento a ser desenvolvido em todas as suas etapas como se fossem *softwares* autônomos, aplicando modelos como o Cascata ou o Evolucionário.

A principal diferença reside na forma como os incrementos são integrados. Essa integração pode ser realizada por meio de interfaces (troca de mensagens) ou por uma integração direta, caso os componentes façam parte de um mesmo *software* e compartilhem uma base de dados comum.

# Modelo Espiral

O modelo Espiral assemelha-se ao modelo Cascata, mas se distingue pela forma como ilustra a evolução do Sistema, permitindo regressões que evidenciam uma espiral de crescimento do valor do Projeto, uma curva ascendente de conhecimento e progresso nas revisões.

Esse modelo é muito comumente utilizado de forma errônea como o Modelo Casca-  
ta, por sua semelhança, mas, como dito, ele representa de forma gráfica o avanço do  
Projeto e seu custo.

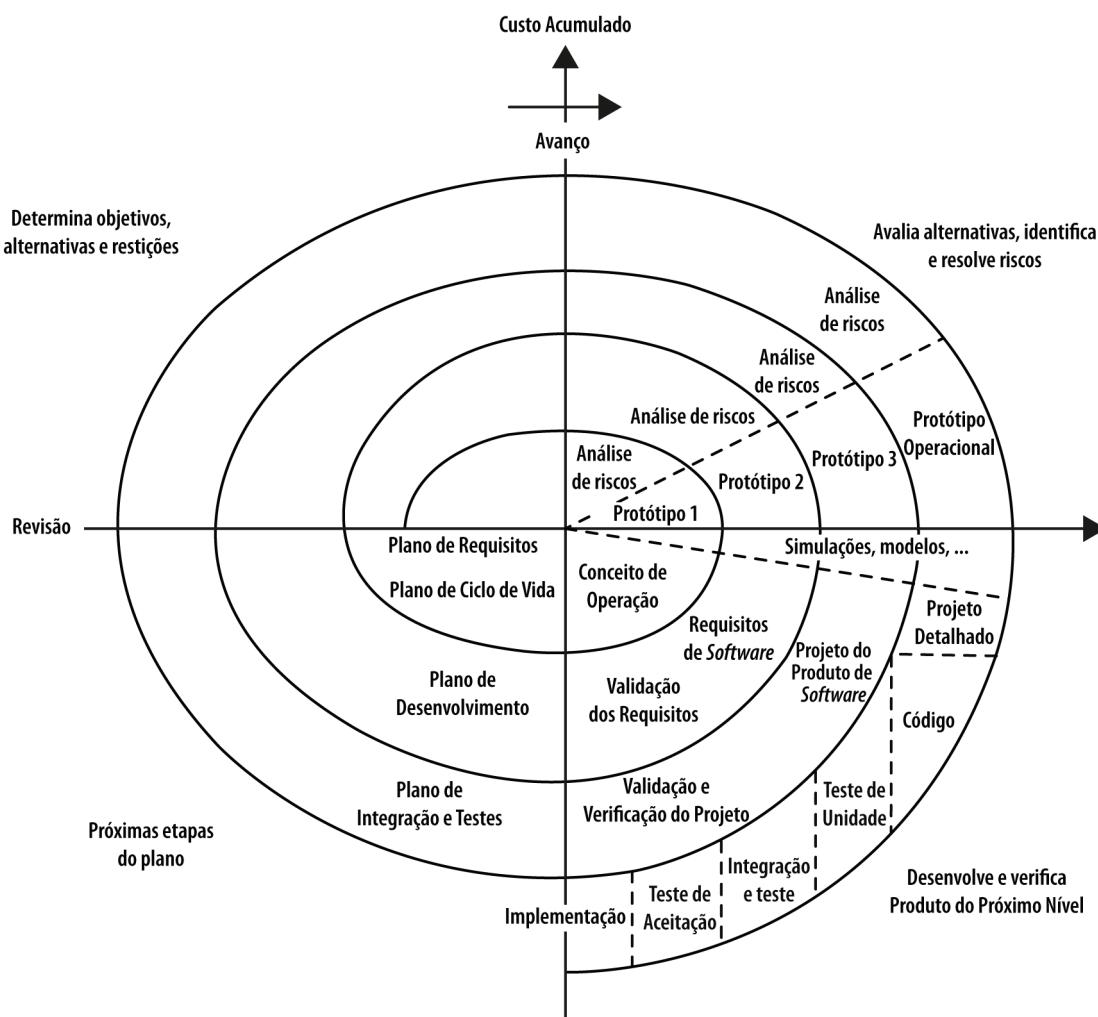


Figura 8 – Modelo Espiral

Fonte: Adaptada de SOMMERVILLE, 2007

**#ParaTodosVerem:** imagem de um Diagrama Espiral que representa as etapas de um Processo de Desenvolvimento, provavelmente relacionado à criação e à implementação de um Projeto, Produto ou Sistema. O diagrama é composto por várias camadas espirais que se estendem do centro para a extremidade externa, cada uma representando uma fase do processo e se expandindo conforme o custo acumulado e o avanço do projeto aumentam. Cada camada da espiral contém várias etapas e processos: 1. No centro, começando pela fase de "Determina objetivos alternativas e restrições"; 2. Seguida por "Análise de Riscos", que parece ser repetido várias vezes à

medida que o projeto avança para as fases externas; 3. As etapas do desenvolvimento do produto ou projeto são sequenciadas em camadas concêntricas com processos como “Plano de Requisitos”, “Conceito de Operação”, “Projeto Preliminar”, “Projeto Detalhado”, “Desenvolve e verifica Produto do Próximo Nível”, entre outros; 4. O processo de “Revisão” é destacado em uma camada, indicando uma etapa de checagem ou controle de qualidade; 5. A espiral continua em direção à superfície externa com etapas adicionais como “Plano de Desenvolvimento”, “Protótipo”, “Simulações, modelos...”, “Validação e Verificação”, e “Implementação”; 6. O final do processo mostra termos como “Teste de Integração”, “Teste de Aceitação”, e “Operação e Manutenção”. Na parte superior da imagem, sobre as camadas da espiral, estão escritos os termos “CUSTO ACUMULADO” e “AVANÇO”, indicando que, à medida que o Projeto avança pelas fases, o custo também aumenta. Há uma seta grande apontando para a direita, indicando a direção de progressão do Projeto, conforme avança no ciclo espiral. A figura visualiza o modelo espiral de desenvolvimento, que é uma abordagem iterativa e progressiva. Fim da descrição.

Para finalizar, é importante saber que cada modelo de processo de *software* é ideal para cada situação. Não é exatamente apropriado afirmar que um modelo é melhor que outro, muito menos aplicável sempre. Por exemplo: o Modelo Cascata para softwares em que são simples e conhecidos os resultados tanto pelo cliente quanto pelo desenvolvedor (Empresa). O modelo evolucionário mais adequado para desenvolvimento de Sistemas que podem funcionar com módulos e podem ser desenvolvidos ao longo do uso dele. O Throwaway se adapta ao desenvolvimento de Sistemas tipicamente em que tanto o cliente quanto o desenvolvedor não conhecem exatamente o resultado. Por fim, o modelo espiral se adapta ao desenvolvimento de Sistemas complexos.

# Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:



Site

**SWEBOK V3**

<https://bit.ly/3Th0Fx8>



*No Silver Bullet: Essence and Accidents of Software Engineering*

<https://bit.ly/4c9VRmD>

**Estudo Comparativo sobre as Técnicas de Elicitação de Requisitos do Software**

<https://bit.ly/3SWzat6>

**Manifesto Ágil**

<https://bit.ly/49U6rMw>

# Referências

PRESSMAN, R. S. **Engenharia Web**. Rio de Janeiro: LTC, 2009.

RUMBAUGH, J.; BRAHA, M. **Modelagem e projetos baseados em objetos com UML**. São Paulo: Elsevier, 2006.

SOMMERVILLE, I. **Engenharia de Software**. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.





**Cruzeiro do Sul**  
Educacional