

## **Aula 1: Processo de desenvolvimento de software**



## **Apresentação**

Nesta aula, iremos definir o conceito de software, quais as suas características e aplicações. Veremos, ainda, o fluxo de informação para a geração de conhecimento.

## **Objetivos**

- Definir o que é Software;
- Identificar as aplicações do Software;
- Compreender os fluxo de dados em um sistema de informação.

# O software

## O que é software?

É um conjunto integrado de programas de Computador.

## O que é um programa de computador?

É um conjunto de instruções que descreve, passo a passo, uma tarefa que deve ser realizada pelo hardware.

Exemplo: Software de Gestão Escolar = conjunto de programas que executam um conjunto de tarefas necessárias a gestão de uma escola.

Dentro do Software de Gestão Escolar, temos o programa que calcula a média dos alunos, com base nas notas informadas das provas.

### Software:



```
$(window).on('resize', function() {
    gridHeights();
});
$(window).resize(function() {
    gridHeights();
});
SC'.close-submenu').click(function(e) {
    $(this).closest('.close-submenu').remove();
    return false;
});
$('#contact-form').ajaxForm({
    success: function(data, status, xhr, form) {
        alert(data.message);
        console.log(form);
        if (data.status == 'success') {
            $(form).resetForm();
        }
    }
});
$('.squares').isotope({
    // options
    itemSelector: 'li',
    masonry: { fitRows: true }
});
```



Código de programação (Fonte:  
Sabrisy/Shutterstock).

Para o desenvolvimento dos programas que integram o software precisamos de uma linguagem de programação, que define as instruções e a forma de relacioná-las.

A primeira linguagem de programação gerava programas, em código de máquina (sequência de 0 e 1), mas era muito difícil para o programador que teria que escrever um código binário (0 ou 1), memorizar as instruções em sequências de 0 e 1.

A partir daí, diferentes linguagens de programação foram criadas, sempre almejando conceder poder e facilidade ao programador. Eram as linguagens de programação de alto nível (receberam esse nome por serem bem próximas da linguagem natural do homem).

Mas o Hardware somente entende a linguagem binária, assim sendo, o programa escrito em linguagem de alto nível precisa passar por um processo adicional, que converta a linguagem de alto nível em linguagem de máquina para ser compreendida e executada pelo hardware.



Linguagem de programação. (Fonte: Shutterstock).

## Linguagem de Programação

Dentre as classificações para as linguagens de programação, a que classifica por paradigmas, avalia como as instruções são compostas e organizadas para formar o programa. Os paradigmas com maior variedade de linguagens de programação são:

### Imperativo

Elementos de código em formato de blocos que se interligam através de três tipos de instrução, da chamada Programação Estruturada.

## Sequência

As instruções são organizadas de forma sequencial (tarefa 1 finaliza, entra tarefa 2).

## Seleção

Onde as instruções podem ser executados baseadas na avaliação de uma condição (IF (Condição=V) THEN Sequencia1 ELSE Sequencia2

## Iteração

Onde as instruções podem ser repetitivas até uma condição ser atingida.

## **Orientada a Objeto (OO)**

Elementos de código em formato de objetos que se interligam.

Um objeto é composto de atributos (dados) e métodos (ações).

Os objetos são agrupados em classes.

Exemplo: Classe ALUNO.

- Atributos: Nome, Matricula, Telefone, endereço
- Métodos: Incluir Aluno, Matricular em curso, Incluir Disciplina

## Classe

Tipo de Objeto.

## Atributos

Variáveis que estão dentro de cada objeto da classe.

## Método

Ação que a classe pode realizar.

# Classificação do Software

Além da linguagem de programação, o software também pode ser classificado como:

## **Software de básico**

São softwares que permitem a operação e programação de computador, como as linguagens de programação e o sistema operacional.

O Sistema Operacional é um software que gerencia o uso do computador/dispositivos móveis, facilitando a vida do usuário e garantindo o melhor funcionamento possível da máquina. O Sistema operacional gerencia o uso do processador, da memória, das unidades de disco (fixa e removível) e todos os periféricos (impressoras, scanners, e etc).

Exemplos de sistemas operacionais: Windows, MacOs, Ios, Android

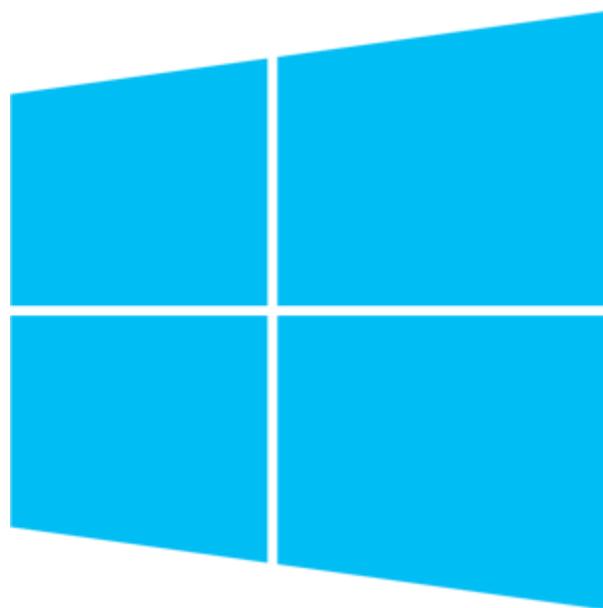


## Monotarefa

Executa somente um processo de cada vez.

### Monousuário

Somente é permitida a utilização de um usuário de cada vez.



## Multitarefa

Os processos são compartilhados e enfileirados a espera do processador. É distribuído de modo que pareça ser executado simultaneamente.

## **Multiprocessamento**

Distribui para mais de um processador.

## **Multiusuário**

Vários usuários utilizam ao mesmo tempo.

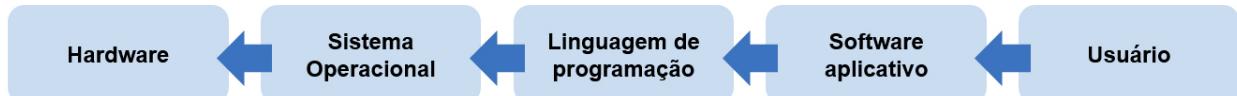
## **Software de aplicativo**

O software aplicativo, como diz o nome, revela alguma utilidade (aplicativo) ao usuário, como por exemplo: editores de texto, planilhas eletrônicas, folha de pagamento, contas a pagar, aplicativos diversos de celulares e etc



A imagem abaixo mostra o fluxo do conjunto de software, básico (sistema operacional e linguagem de programação) e aplicativo, à serviço do usuário.

Um conjunto de camadas, de forma a tornar o computador (hardware e software) uma ferramenta de trabalho ao usuário





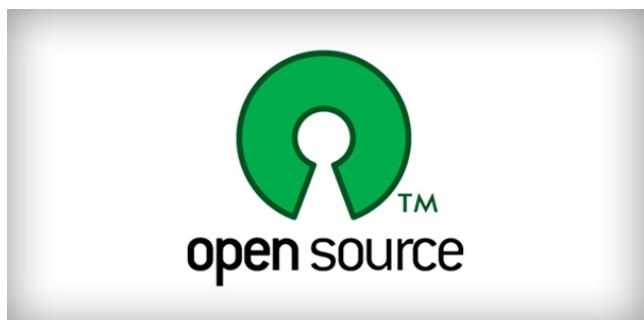
(Fonte: Raw Pixel/Shutterstock).

# Características e aplicações do software

O software pode ser classificado de acordo com a sua forma de cópia e distribuição (licença de uso). Em geral, o software pode ser:



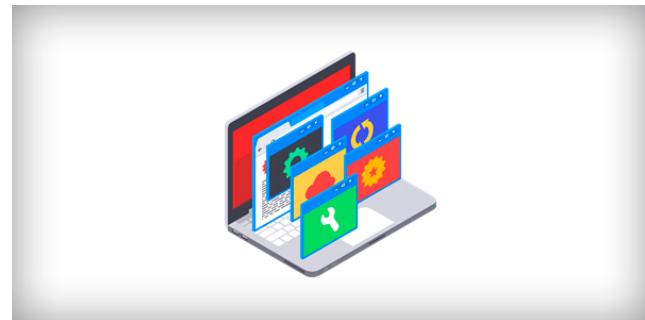
## 1. Software Gratuito ou Freeware



## 2. Software Livre



### 3. Comercial



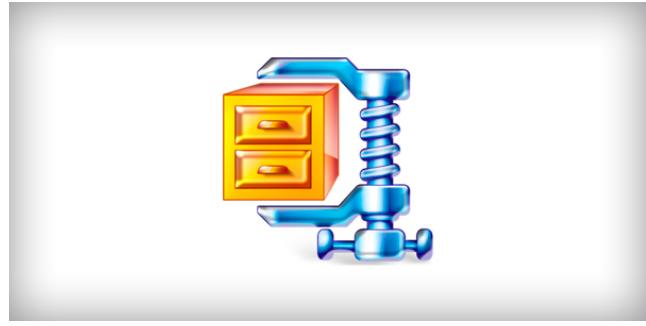
### 4. Adware



### 5. Demo



### 6. Trial



## 7. Shareware

---

### 01

Programa de computador cujo uso não implica o pagamento de licença de uso. O Freeware pode ser copiado e distribuído gratuitamente.

O Freeware pode ser utilizado sem pagar, mas o código fonte não é disponibilizado, logo o freeware não pode ser modificado.

Assim sendo o freeware só pode ser usado da forma como é disponibilizado.

---

### 02

Programa de computador cuja utilização, cópia e distribuição não possui restrição. É comum o código fonte estar disponível para manuseá-lo, e dessa forma o software pode ser modificado, por quem desejar, sem que seja necessária a permissão ou que se pague ao autor.

---

### 03

Programa onde deve-se pagar um valor para sua aquisição e/ou uso.

---

## **04**

Programa de computador que executa automaticamente algum tipo de publicidade após sua instalação ou durante sua utilização.

---

## **05**

Fração de um programa. Funciona como material promocional para dar a oportunidade do produto ser avaliado. É restrito de suas funcionalidades apenas para teste.

Por exemplo sistema de contas a pagar que permite apenas 5 movimentações no mês.

---

## **06**

Programa semelhante ao demo, mas com funcionalidades disponíveis por tempo determinado.

Após o prazo de uso do software Trial é preciso adquirir a licença.

---

## **07**

Programa de computador que possui limitações de tempo e/ou funcionalidades. Ao final do tempo estabelecido, o programa pode requisitar o pagamento para uso do software completo ou pode continuar rodando sem todas as suas funcionalidades ou, ainda, interromper o seu uso.

---

# O processo de desenvolvimento de software

O desenvolvimento de software é um processo com grande dependência da subjetividade humana, de difícil automação.

A partir de meado dos anos 80, após a crise, o software passou a ser desenvolvimento de forma mais ordenada, usando conceitos da engenharia, quando foi cunhado o tempo Engenharia de Software.

A atividade de desenvolvimento de software passa ser vista como um processo, organizado em fases, cada qual com uma finalidade, objetivando um software de maior qualidade e um processo de desenvolvimento menos subjetivo.

O processo de desenvolvimento do software, deve estabelecer:

- Quais as fases do processo?
- Qual a finalidade de cada fase?
- Ordem e ligação entre as fases
- Artefatos (modelos) / produtos gerados pela fase
- Documentação de cada fase

A imagem mostra as Fases mais comuns, presentes em alguns dos principais modelos de processos de desenvolvimento de software



**Concepção:** fase onde o sistema é concebido e avaliada a sua viabilidade (técnica, econômica, tempo). Caso viável o desenvolvimento segue nas fases seguintes e caso contrário é interrompido.

**Requisitos:** Em muitos modelos de processos de desenvolvimento, a fase de requisitos pode estar junta com a de Analise, mas a finalidade é identificar as necessidades dos usuários para definir os requisitos do sistema.

**Análise:** fase de estudo, onde é definido O QUE o sistema deve fazer, independente da tecnologia que venha a ser usada. Define-se, dentre outras coisas as funcionalidades que o sistema precisa ter.

**Projeto:** onde define-se as tecnologias a serem usadas: linguagem de programação e banco de dados. É também a fase de definição da arquitetura do software e seus respectivos elementos. E fase do COMO fazer o software.

**Codificação:** é a escrita de cada programa que vai compor o sistema, na linguagem de programação selecionada. Depende tecnicamente da qualidade do programador para gerar códigos inteligíveis e de fácil manutenção.

**Testes:** Verificação de erros no sistema e apuração se o mesmo executa as ações previstas e necessárias a seus usuários. Abrange um conjunto de testes em diferentes momentos da fase de codificação. Deve-se testar, inicialmente, cada programa separadamente e depois o conjunto integrado de programas.

**Homologação:** fase onde os usuários atestam que o software atende (ou não) as suas necessidades, liberando-os (ou não) para uso.

**Implantação:** fase onde o sistema é posto em uso, no ambiente do usuário, o que requer instalação dos softwares , treinamento a usuários e acompanhamento do uso por um período de tempo (acordado previamente, em contrato, preferencialmente).

**Manutenção:** uma vez implantado, o sistema precisa sobreviver ao longo dos anos, adequando-se as mudanças da empresa e do contexto onde a mesma esta inserida.

Características do processo de desenvolvimento de software

- Dependente do componente humano
- Pouca automação no processo
- Pouca aplicação da visão de projeto
- Pressão dos usuários: rapidez a baixo custo
- Dificuldade de comunicação entre os membros da equipe

Problemas:

- Instabilidade dos requisitos, que mudam com frequência.
- O software hoje é grande, complexo e com interface para outros sistemas.
- A gestão do processo está mais complexa, em função de agregar mais recursos (humanos e tempo).

Consequências aos problemas acima.

- Prazos extrapolam
- Custos extrapolam

- Software sem qualidade

# Sistema de informação

**Sistema** = Conjunto de partes, independentes, cada qual com seu objetivo e colaborando por um objetivo comum.

**Informação** = Dados (fatos isolados) agrupados e relacionados (processados), com sentido lógico.

-Dados: chq 1235 de 1.250,00, chq 1236 de 750,00

-Dado: saldo anterior é 5.000,00

-Informação: saldo atual é 3.000,00

**Sistema de Informação** = Conjunto de elementos inter-relacionados que coleta (**entrada**), manipula (**processamento**), armazena e dissemina (**saída**)

## Referências

---

BELL, Suzanne. **Forense Chemistry**. 2. ed. Edinburgh, England: Pearson, 2014, cap. 14.

HOLLER, F. James; SKOOG, Douglas A.; CROUCH, Stanley R. **Princípios de Análise Instrumental**. 6. ed. Porto Alegre: Bookman, 2009, cap. 6.

NETTO, Amilcar; ESPÍNDULA, Alberi. **Manual de atendimento a locais de morte violenta**. 2. ed. Campinas, SP: Millennium, 2016, cap. 3 e 20.

SANTIAGO, Elizeu. **Criminalística Comentada**. 1. ed. Campinas, SP: Millennium, 2014, módulos 18, 28, 45.

TOCCHETTO, Domingos; STUMVOLL, Vitor. **Criminalística**. 6. ed. Campinas, SP: Millennium, 2014, cap. 3.

## Próximos Passos

---

- Requisitos que devem ser levantados para viabilizar o desenvolvimento de um software.

## **Explore Mais**

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## **Aula 2: Processo de desenvolvimento de software**



## **Apresentação**

Nesta aula, iremos definir o conceito de requisito para o processo de desenvolvimento de software.

O entendimento das atividades da chamada engenharia de requisitos permite reduzir os erros decorrentes dessa fase que corresponde ao inicio do ciclo de vida do processo de desenvolvimento de software.

---

## **Objetivos**

- Descrever as atividades para análise dos requisitos no processo de desenvolvimento de software;
- Desenvolver técnicas para eliciar e analisar requisitos;
- Compreender o gerenciamento de requisitos.

# Atividades para análise de requisitos

Todo projeto, em sua fase inicial, deve ser submetido a um estudo de viabilidade, cujo objetivo, como o próprio nome já diz, é saber se vale a pena desenvolver a ideia sob o ponto de vista operacional, técnico, econômico e organizacional.

O estudo deve oferecer base para ajudar nessa decisão, questionando:

**O projeto/produto pode ser feito?**

**O projeto/produto beneficiará os clientes interessados?**

**Existe uma alternativa?**



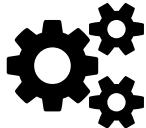
Reunião (Fonte: Raw Pixel/Shutterstock).

O estudo de viabilidade envolve quatro áreas distintas em sua análise:



# **TÉCNICA**

Visa a atender os requisitos técnicos do produto a ser desenvolvido. O levantamento deve estar relacionado com a tecnologia existente no processo de desenvolvimento.



# **OPERACIONAL**

Visa atender os requisitos para a aceitação do produto ou problema apresentado. O levantamento deve estar relacionado com a aceitação da solução proposta, e como os agentes se sentirão em relação a ela.



# **CRONOGRAMA**

Visa a atender os requisitos de tempo para os prazos estabelecidos. O levantamento deve estar baseado na viabilidade técnica em relação ao prazo estipulado. Prazos obrigatórios são mais difíceis de serem negociados.



# **ECONÔMICA**

Visa a atender os requisitos financeiros do projeto/produto. Considerada a mais critica, consiste em julgar se o projeto será deficitário ou se os custos de sua implementação não terão os benefícios desejados.

## A Fase Econômica também é chamada Análise de Custo-Benefício.

Ela deve contemplar custos de:

Operação	Desenvolvimento do projeto
Custos fixo e contínuos. Ex.: pessoal, manutenção, energia.	Custos que ocorrem somente uma vez. Ex: aquisição de novos softwares; instalação; atualização.

Análise de ROI (*Return On Investment*)

Percentual que mede a relação entre quanto se ganhou e quanto se investiu.

**ROI = (total de lucro – total custo) / total de custo**

**Quanto maior for a taxa, melhor será o ROI.**

## Necessidades prévias

### REQUISITO

É uma condição ou necessidade para resolver um problema ou alcançar um objetivo. Também pode ser estar presente em um sistema a fim de satisfazer uma condição, um contrato, um padrão, ou uma especificação devida.

### REQUISITO DO USUÁRIO

Definições sobre a função do sistema e as restrições sob os quais ele deve operar. O formato é em linguagem comum, visando ao entendimento do cliente/usuário.

## **REQUISITOS DO SISTEMA**

Definição estruturada e detalhada do serviço que será feito no sistema/produto. O formato é em Contrato de Prestação de Serviço entre o cliente e o fornecedor.

## **REQUISITOS FUNCIONAIS**

Descrevem as funcionalidades do sistema. Estão diretamente ligados às especificações da tecnologia envolvida, do perfil do usuário, do tipo do sistema.

Exemplos:

[RF 0023] Usuário não pode acessar o banco de dados financeiro.

[RF 0059] Sistema deve oferecer opção para o usuário escrever observação nos documentos.

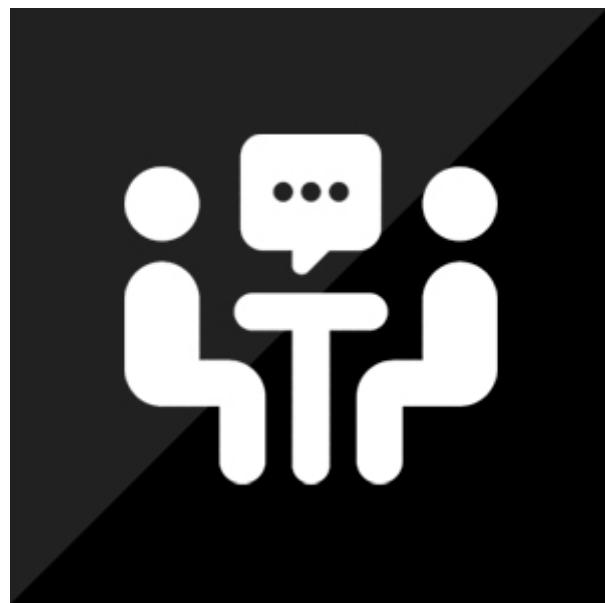
## **REQUISITOS NÃO FUNCIONAIS**

### **Algumas propriedades e suas medições**

Velocidade	Transações / segundos
Tamanho	Mbytes
Confiabilidade	Tempo médio de falhas
Facilidade de uso	Treinamento

Antes de continuar seus estudos, saiba mais sobre [requisitos não funcionais <galeria/aula2/anexo/requisitos-nao-funcionais.pdf>](#)

# Técnicas de elicitação



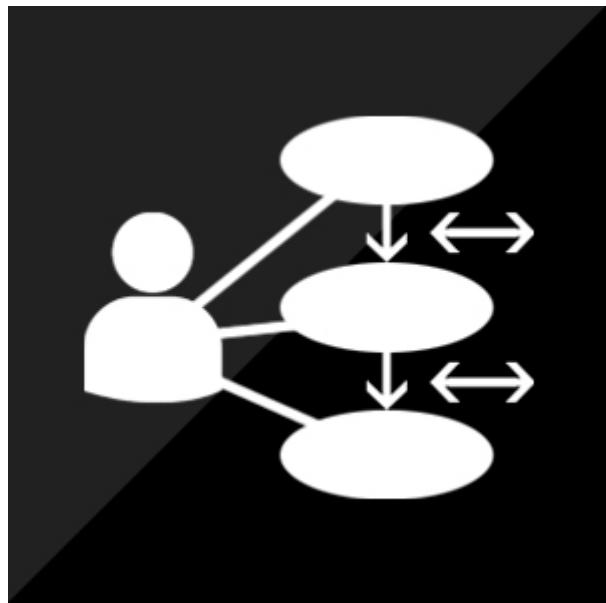
## Entrevista

Utilização na análise de problema e na engenharia de requisitos com o objetivo de entender as perspectivas do cliente/usuário. Entender quem são os agentes e quais as necessidades, o problema e a solução.



## **Questionários**

Forma de utilização que faz perguntas referentes ao sistema. Utilização de hipóteses para as relevâncias. Podem ser utilizados após a entrevista.



## **Casos de uso**

Identificação dos agentes que atuam no sistema; das interfaces que o sistema/produto possuirá; validação de pré-requisitos. Representação visual ao invés de textual.



## ***Brainstorm***

Ou tempestade de ideias, faz o levantamento de ideias, em que cada uma sugerida pode combinar na propositura de uma nova. Atividade de livre imaginação que deve ser tratada sem críticas ou debates.

---

## **Referências**

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

---

## **Próximos Passos**

- A etapa de análise onde se trabalha e modela os requisitos a fim de se obter uma estrutura para auxiliar no desenho da solução.

---

## **Explore Mais**

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## **Aula 3: Atividade de análise no processo de desenvolvimento de softwares**



## **Apresentação**

Nesta aula, iremos definir o conceito de análise para o processo de desenvolvimento de software.

A fase de análise tem como objetivo fazer uma modelagem dos agentes, separando-os em objetos, classes e atributos.

A análise pode ser estrutural ou comportamental.

---

## **Objetivos**

- Conhecer as atividades de análise de desenvolvimento de software;
- Entender os relacionamentos dos objetos;
- Modelar os relacionamentos dos objetos.

# Conceito de Modelagem

## Modelagem

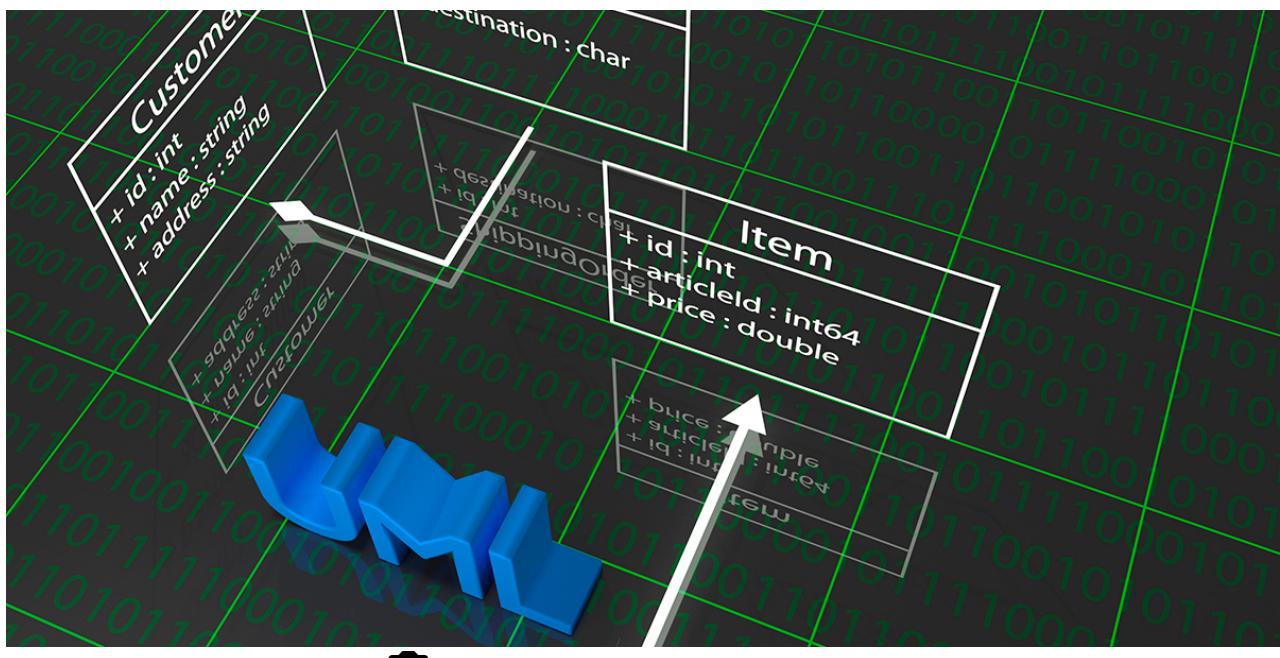
Serve para verificar a qualidade dos requisitos, estudados na aula anterior, que se tornarão precisos e detalhados o suficiente para as atividades do próximo passo no processo de desenvolvimento de software.

## Análise

Atividade que utiliza o conceito de orientação a objeto, utilizando a UML como notação. Tem como objetivo modelar o problema, não a solução.

## UML

Unified Modeling Language, linguagem de modelagem unificada, utilizada em engenharia de software para visualizar o desenho do sistema e a intercomunicação entre objetos.



UML (Fonte: Ton Snoei/Shutterstock).

# Objeto e classe

## Objeto

Estrutura de dados encapsulada por procedimentos. Essa estrutura são os atributos e operações.

## **Classe**

Conjunto de objetos similares agrupados em que a etapa de análise está mais voltada para sua realização.

Vejamos um exemplo para entender melhor.

**Classe:** Pessoa.

**Objetos:** Pessoas.



📷 Mosaico de pessoas (Fonte: Etraveler/Shutterstock).

## **Tipos de análise**

### **Análise Estrutural**



Tem como objetivo modelar aspectos estáticos de um problema, utilizando o modelo orientado a objeto. É utilizada em conjunto com detalhamento de requisitos para visualizar e fornecer base para identificar soluções para os

requisitos apresentados.

Vejamos as atividades dentro da análise estruturada:

### **Identificação de classes**

Identificar quais são as classes chaves. Fazer o levantamento com base em suas responsabilidades e colaborações. Utiliza-se em larga escala o cartão CRC (*Class-Responsability-Collaborator*).

### **Organização das classes**

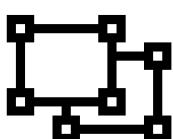
Organizar as classes em três tipos:

<b>Entidade</b>
Representa conceitos do domínio do problema herdada dos modelos de negócio.
<b>Fronteira</b>
Representa interfaces externas que estão dentro do produto, como interface de usuário e conexão com outros sistemas. Facilita o desenho das interfaces.
<b>Controle</b>
Organização que não pertence à entidade e nem à fronteira. Normalmente é associada a um caso de uso.

### **Identificação dos relacionamentos**

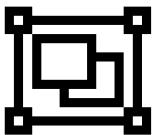
Ajuda a filtrar e refinar as classes.

Pode se por:



## **1. Associação**

Indica a relação entre duas classes em que o objeto de uma classe consegue obter informações da outra a que foi associado.



## 2. Agregação

Indica um associação, mas com a classe se apossando das informações de um objeto da outra.

### Identificação dos atributos

A cada classe é atribuída uma característica responsável por tomar alguma ação.

### Análise comportamental

Aplicada depois que os requisitos forem detalhados, validando-os e indicando as dificuldades de implementação no plano de conceito.

### Diagrama de interação

Mensagens que são trocadas, ao longo do tempo, para execução de alguma tarefa.

Mensagens e Operações: representam um mecanismo de interação, ou seja, um objeto só poderá receber uma mensagem invocada por uma classe.

A mensagem tem as seguintes partes:

<b>Receptor</b>	<b>Operação</b>	<b>Parâmetro</b>
-----------------	-----------------	------------------

**Interação:** como as mensagens trafegarão para a execução de uma tarefa.

**Diagrama de sequência:** ordem temporal das ações que serão executadas.

### Identificação das operações

Todas as mensagens devem se mapeadas para executarem alguma operação.  
Podem ser: Incluir, Alterar, Excluir, dentre outras.

**Podem ser:** Incluir, Alterar, Excluir, dentre outras.

# Notas

## Referências

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

- Etapa de desenho onde se trabalha e modela os requisitos para se obter uma estrutura para auxiliar no desenho da solução.

## Explore Mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## Aula 4: O desenho no processo de desenvolvimento de software



## Apresentação

Nesta aula, iremos definir o conceito de desenho para o processo de desenvolvimento de software.

A fase de desenho tem como objetivo modelar o sistema, atendendo os requisitos levantados na fase de análise, e prepará-los para a implementação.

O desenho do produto ou solução mostra como deve ser implementado, mas não envolve qual o tipo de tecnologia específica necessita para fazê-lo.

---

## Objetivos

- Conhecer as atividades de desenho ou arquitetura no processo de desenvolvimento de *software*;
- Diferenciar os modelos de desenhos para as suas atividades;
- Entender as necessidades de desenhar a solução analisando os requisitos.

# Problema vs. Solução



## Problema

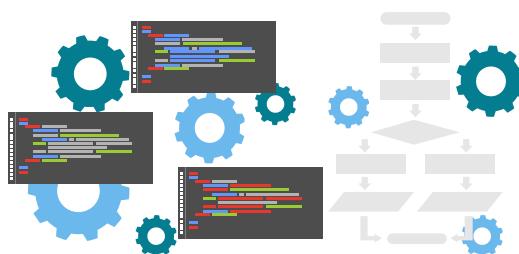
Através do levantamento de informações na fase de análise de requisitos, define-se um problema ou meta a ser alcançado.



## Solução

Após levantamento, uma solução deverá ser escolhida dentre várias possíveis, isto é o papel do arquiteto de software. A documentação do desenho explicita a solução que será tomada para resolução do problema.

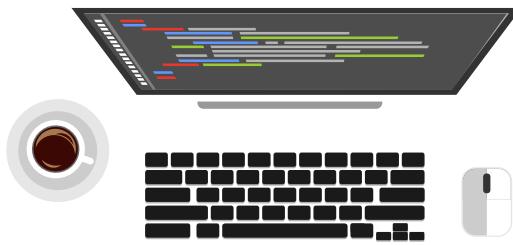
## Modelos de desenho



## Desenho interno

É a maneira como o sistema interage com outros produtos ou sistemas. Podem conter parte físicas, lógicas, interconexões com outros sistemas e produtos, interna ou externamente.





## Desenho externo

Visão que os **usuários** terão da solução ou produto e a forma com que eles interagirão.

O nível de abstração e agregação dos elementos dos sistemas podem ser:

### Nível estratégico ou desenho arquitetônico

É o corpo da arquitetura do sistema a ser implementado. Com base nesse desenho, já se pode saber se o sistema atenderá aos requisitos e aos custos relacionados do projeto.

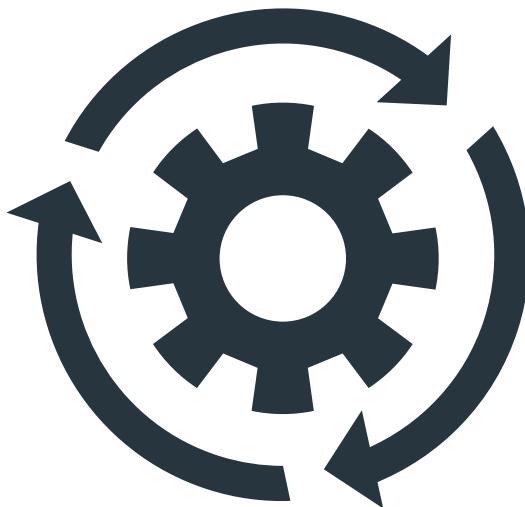
### Nível tático ou desenho lógico

É a aplicação das decisões tomadas no nível estratégico. A solução contemplará a reutilização, ou não, de componentes, que serão desenvolvidos para ele, buscando satisfazer os requisitos do produto.

### Nível operacional ou desenho detalhado

É o comportamento de cada componente. É desenvolvido em conjunto com a documentação voltada para usuários, no caso de desenho externo, ou documentação do código do programa, no caso de desenho interno.

# Reutilização



Nesta fase, é comum se fazer uso de processos que já foram definidos, usados e testados em outros projetos ou produtos, evitando o retrabalho.



## Atenção

O processo de reutilização visa à redução do desperdício de tempo e, consequentemente, dinheiro, visto que, podemos aplicar o conceito de reutilização em diversos processos em todas as fases do desenvolvimento do software, gerando melhora significativa na qualidade e na produtividade.



1

## Código

Reutilização de parte ou todo ó código fonte de módulos do sistema. Só é possível quando na fase de construção do código aplicou-se corretamente conceitos como o de coesão, acoplamento.



2

## Reutilização de objeto

Bibliotecas e classes fundamentais.



3

## Reutilização de plataforma

Reutilização de Plataforma - Camada de arquitetura, consiste na colaboração de ferramentas para gerenciar e automatizar o ciclo de vida de desenvolvimento de software, com a utilização de "templates" de processos.



4

## Desenho

Aproveitamento de ideias para solução de problemas, são comumente encontradas em padrões, frameworks e arquiteturas, abordagens que permitem a reutilização ao nível de desenho.



5

## Reutilização de classe

Módulo de código binário, mecanismo baseado em: Composição ("tem um") e Herança ou derivação ("é um").

## Referências

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

Etapa de codificação onde será apresentada a parte de desenvolvimento do código baseado nos modelos e desenhos apresentados.

## Explore Mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## **Aula 5: As atividades de teste no processo de desenvolvimento de software**



### **Apresentação**

Nesta aula, iremos definir o conceito de teste para o processo de desenvolvimento de *software*.

A fase de teste tem como objetivo detectar possíveis defeitos ou erros que possam surgir na fase de implementação. Nessa fase, de testes, deve-se coletar os resultados e analisá-los e consertá-los antes de sua implantação.

Essa fase é essencial para aumentar a qualidade do produto ou sistema em que será implantado.

---

### **Objetivos**

- Conhecer as atividades de teste no processo de desenvolvimento de *software*;
- Entender as necessidades da etapa de teste na melhora da qualidade do sistema;
- Analisar os diversos tipos de teste para sistema ou produto.

# Testes de Software



● Teste de software. (Fonte: alphaspirit / Shutterstock)

## Conceito de teste para o processo de desenvolvimento de software

Teste de Software é uma ação que faz parte do processo de desenvolvimento de software, e tem como principal objetivo o de revelar falhas para que sejam corrigidas até que o produto final atinja a qualidade desejada ou acordada.

Os analistas de testes, técnicos de testes ou homologadores, são responsáveis para realizar uma bateria de testes de diferentes naturezas e propósitos, envolvendo não apenas os testes funcionais da aplicação, mas diversas outras atividades como:

- Avaliação da especificação de requisitos,
- Avaliação de projeto técnico,
- Verificações em outros documentos,
- Testes de performance e capacidade,
- Avaliação de interface,
- Dentre outros.

“

**“Quanto mais cedo, defeitos forem encontrados antes da implementação do sistema, o custo de correção é menor em relação ao encontrado na fase de produção”**

---

(Regra de Myers)

De acordo com o padrão IEEE número 610.12-1990, podemos classificar:

**Defeito** – passo, processo ou definição de dados incorreto, por exemplo, uma instrução ou comando incorreto.

**Engano** – ação humana, por exemplo, tomada pelo programador, que produz um resultado incorreto.

**Erro** – diferença entre o valor obtido e o valor esperado. Qualquer resultado incorreto ou inesperado na execução do programa.

**Falha** – produção de uma saída incorreta com relação à especificação.



---

## CAUSA

Engano

Defeito

Erro



---

## CONSEQUÊNCIA

Falha

**Teste de Software** é um processo que faz parte do desenvolvimento de software, e tem como principal objetivo revelar falhas/bugs para que sejam corrigidas até que o produto final atinja a qualidade desejada.



### Teste

Processo definido com intenção de encontrar um erro.



### Objetivo

Encontrar um erro que ainda não foi descoberto. Um teste bem sucedido corresponde à descoberta de um erro não previsto.



### Critério

Definição de uma métrica que, após análise do comportamento do sistema, atenda o critério.



4

## Procedimento

Conjunto de instruções para a realização de testes.



5

## “Script” de teste

É uma representação definida de um procedimento teste.

# Processo de Teste de Software

O Processo de Teste de Software divide-se em três partes principais e fundamentais:

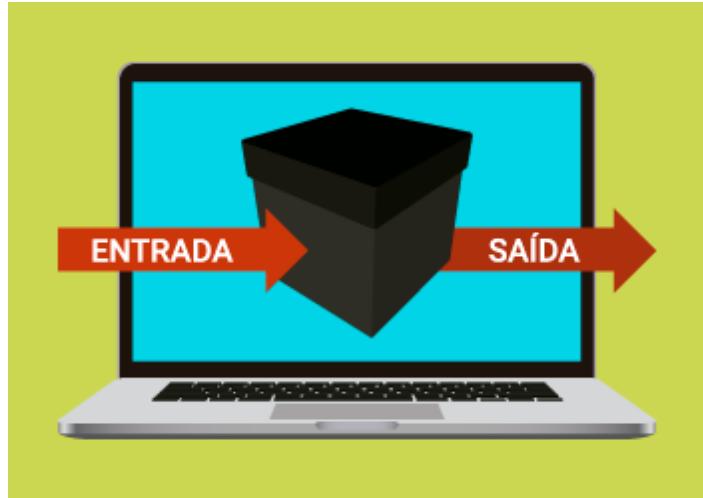
- **Planejamento dos testes:** “Garantir que os testes sejam preparados antes do fim da implementação do produto”.
- **Execução dos Testes:** “Executar os casos e procedimentos de teste especificados e comparar os resultados esperados e obtidos, registrando esses resultados”.
- **Controle dos testes:** “Garantir que os testes planejados sejam executados corretamente e seus resultados possam ser registrados através da sua monitoração constante”.

Vejamos alguns tipos de testes:



## **Testes de sistemas**

Neste teste, o sistema é analisado como um todo, ou seja todos os seus componentes, para validar a execução das suas funções acompanhando cenários chamados de casos de teste e verificar se estão em conformidade com os requisitos anteriormente definidos. Este teste deve ser feito por uma equipe independente e diferente daquela que desenvolveu o sistema.



## **Teste caixa preta**

Neste teste o objetivo é testar todas as entradas e saídas desejadas, mediante uma determinada entrada definida de dados. Aqui não se está preocupado com o código, cada saída indesejada é vista como um erro. Os mecanismos internos do sistema não são levados em conta



### **Teste caixa branca**

Neste caso o objetivo principal é testar o código dos componentes do sistema, quanto a sua estrutura e construção. Os mecanismos internos do sistema serão analisados e suas representações lógicas também. Este teste não exclui a necessidade do Teste Caixa Preta, uma vez que o funcionamento interno do sistema ou produto pode corresponder logicamente, podendo produzir uma saída diferente da esperada. É comum se encontrar partes do código que nunca foram testadas.



### **Testes de Segurança**

Dada as circunstâncias atuais, onde cada vez mais a segurança dos sistemas é posta à prova, o Teste de Segurança tem como meta garantir que o funcionamento da aplicação esteja exatamente como especificado. Verifica também se o software se comporta adequadamente mediante as mais diversas tentativas ilegais de acesso, visando possíveis vulnerabilidades. Para isso, testa se todos os mecanismos de proteção embutidos na aplicação de fato a protegerão de acessos indevidos. Deve-se utilizar os diversos papéis, perfis, permissões, para navegar no sistema.

# Modalidade dos testes

Quanto à utilização do código:

## Testes estáticos

São testes realizados pela análise estática e visual do código fonte, podendo haver um questionário ou alguma ferramenta para acompanhar os testes. Nesse caso, os componentes do software são verificados sem que o produto seja executado. Seja por meio de uma ferramenta automatizada ou dos testes manuais, o principal objetivo dessa técnica é identificar erros de programação, tais como:

- Prática ruins de programação
- Práticas ruins;
- Erros de sintaxe;
- Falhas de segurança.

A análise estática auxilia gestores de TI a identificar todas as linhas de código que foram mal escritas durante a criação de um software. Todos os caminhos de execução, processamento e exibição de valores são examinados. Como consequência, erros mais comuns são descobertos mais rapidamente.

## Testes Dinâmicos

São testes baseados na execução do código do programa, podendo ser encarado como de forma complementar a análise estática. Trabalha, principalmente, com as informações que são inseridas nas rotinas de entrada e saída de dados. Além disso, são verificados itens como:

- O tempo de resposta;
- A performance da aplicação;
- A capacidade do software se adaptar a diferentes ambientes;
- O comportamento funcional.

A análise dinâmica permite que problemas mais sutis sejam identificados. Sem considerarmos o grau de complexidade, as chances de um “bug” passar por uma análise estática e uma análise dinâmica, sem ser rastreado é consideravelmente baixa. Dessa forma, o teste dinâmico consegue dar mais segurança e confiabilidade ao produto final.

E quanto ao objetivo na busca pelo erro:

### **Testes de unidade**

Teste em nível de módulo, componente ou classe. É o teste cujo objetivo é um “pedaço do código”, ou em alguns módulos definidos que representam uma única unidade. A documentação do projeto define a quantidade de módulos a serem testados.

### **Testes de integração**

Teste utilizado para garantir que um ou mais componentes combinados, ou unidades, não contenham erros. Podemos dizer que um teste de integração é composto por diversos testes de unidade.

### **Testes de validação**

Teste realizado após a integração de todos os módulos ou unidades do sistema.

## **Referências**

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

Etapa de codificação onde será apresentada a parte de desenvolvimento do código baseado nos modelos e desenhos apresentados.

## Explore Mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## **Aula 6: A Implementação no Processo de Desenvolvimento de Software**



### **Apresentação:**

Nesta aula, iremos definir o conceito de implementação para o processo de desenvolvimento de software.

A fase de implementação, ou codificação, tem como objetivo escrever o programa em uma linguagem de programação, seguindo normas e diretrizes da empresa à qual o desenvolvedor esteja ligado.

Na fase da implementação, o analista ou desenvolvedor detalha e implementa o que foi definido na etapa de desenho, através de componentes de código de programa e documentação detalhada.

---

### **Objetivos:**

- Conhecer as atividades de implementação no processo de desenvolvimento de software;
- Entender as necessidades de definir uma tecnologia para a transformação do desenho para o projeto em um sistema binário;
- Analisar os diversos tipos de produto e utilizar a linguagem que atenda às necessidades;

# Definições

Nós vimos anteriormente, que **desenho**, é uma das etapas do processo de desenvolvimento de software, lembra?

A **implementação**, é o processo que realiza a transformação do desenho em diversos tipos de componentes de código de programação.



O código de programação pode ser dividido em 3 tipos:

```

    CV_HOUGH_GRADIENT, 2, gray->width,100,200); if(circles->total > 0)
{
    float* p = (float*)cvGetSeqElem( circles, 1 );
    uchar* ptr = cvPtr2D(img, cvRound(p[1]), cvRound(p[0]), NULL);

    double region_size = 7;
    double red_avg = 0;
    double green_avg = 0;
    double blue_avg = 0;

    for(int y=floor(region_size/2); y<ceil(region_size/2); y++)
    {
        uchar* ptrl = (uchar*) (ptr + y * img->widthStep);
        for( int x=floor(region_size/2); x<ceil(region_size/2); x++)
        {
            blue_avg += ptr[3*x];
            green_avg += ptr[3*x+1];
            red_avg += ptr[3*x+2];
        }
    }
    red_avg = red_avg/(region_size*region_size);
    green_avg = green_avg/(region_size*region_size);
    blue_avg = blue_avg/(region_size*region_size);

    bool color = (green_avg-150)*(green_avg-150)<900 && (blue_avg-100)*
    if(color)
    {
        cvCircle( rgbiimg, cvPoint(cvRound(p[0]),cvRound(p[1])),
                  3, CV_RGB(0,255,0), -1, 8, 0 );
        cvCircle( rgbiimg, cvPoint(cvRound(p[0]),cvRound(p[1])),
                  cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );
    }

    if(d = get_actual_depth(cvGet2D(depthimg, cvRound(p[1]), cvRound(
    {
        tempLandmark->detected = true;
        X   = 320.5 - cvRound(p[0]);
        mu  = (240.5 - cvRound(p[1]))*d/FOCAL_LENGTH;
        w   = X*d/FOCAL_LENGTH;
    }

```

## Código Fonte

Conjunto de instruções geradas através de uma linguagem de programação, de maneira lógica e estruturada; após o processo de compilação ou interpretação, transformar-se-á em código objeto.

```

MOV id2 R1
MULT 2 R1
MOV id1 R2
ADD R1 R2
MOV R2 id1

```

## Código Objeto

Resultado da compilação do código fonte.

```
10100110011100000110101  
00011100010101000111100  
1100001111101010101011  
00110100110101010101110  
11000110101000011000111  
0101010100001110011010  
10100111010110011101101  
00110101011010111010101  
0101010001001100011110  
1010101001110010110001
```

## Código Máquina

Sequência binária de ações diretamente direcionadas para o processador da máquina.

Vejamos mais algumas definições:

### Compilador

Programa que faz uma leitura do código fonte, desenvolvido em uma linguagem de alto nível, e transcreve para um novo tipo de linguagem chamada de baixo nível.

### Interpretador

Programa que, além de fazer a leitura do código fonte e transformá-lo em código objeto, efetua a execução do mesmo sequencialmente.

### Linguagem de baixo nível

Linguagem de programação que utiliza a arquitetura do processador para executar as ações. Esta linguagem é a que mais se aproxima dos códigos de execução direta do processador, ou seja, linguagem de máquina.

## Linguagem de alto nível

Comumente chamada de linguagem de programação, esta linguagem se aproxima mais da linguagem humana, ou seja, linguagem com um padrão de entendimento humano bem definido. Para essa linguagem não é levado em consideração a arquitetura do computador, nem as características do processador e seus registradores, visto que, na fase de interpretação ou compilação, esses programas transformarão em linguagem de baixo nível ou de máquina.

# Classificações das linguagens



Linguagem de programação (Fonte: Shutterstock / Abscent)

## Linguagem de primeira geração

Desenvolvida no inicio da era dos computadores, esta linguagem é interpretada pelos microprocessadores. Cada microprocessador possui uma linguagem própria de entendimento, o que pode ocasionar erros de programação em processadores de uma mesma família de fabricantes.

Ex: Linguagem binária. 0 e 1.

## **Linguagem de segunda geração**

Surgida em meados dos anos 50, foi considerada a primeira linguagem de alto nível, visto que era de fácil entendimento e, portanto, considerada mais humana.

Ex: Assembly

## **Linguagem de terceira geração**

Em meados dos anos 80, surgiram com os conceitos de programação estruturada e programação orientada a objetos. Ex: Pascal, Cobol, C, C++

## **Linguagem de quarta geração**

É característica dessa linguagem dar suporte para execução de rotinas auxiliares a linguagens de terceira geração.

Ex: Linguagem de consulta, utilizada para conexão com banco de dados. (SQL)

Uma vez que o desenho será a base da implementação, o processo de documentação de uso do produto passa a ter importância nesta fase, onde a documentação e a programação devem andar lado a lado.



## Referências

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

Etapa de documentação e manutenção do produto no processo de desenvolvimento de *software*.

## Explore mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## **Aula 7: A documentação do sistema de software**



## **Apresentação**

Nesta aula iremos definir o conceito de manutenção para o processo de desenvolvimento de software.

A fase de manutenção, tem como objetivo corrigir os erros que não foram detectados nas fases anteriores, propor melhorias no sistema e prover suporte do sistema que foi desenvolvido.

Nessa fase, é fundamental o acompanhamento do produto desenvolvido, para o suporte e o processo de manutenção, entrando assim em uma nova fase de análise de requisito, ou seja, a fase inicial do processo de desenvolvimento de software.

---

## **Objetivos**

- Conhecer as atividades de documentação, suporte e melhoria no processo de desenvolvimento de software;
- Entender as necessidades de documentar as atividades, os processos e procedimentos durante o desenvolvimento de um sistema;
- Analisar as diversas formas de documentação e sugestões de melhoria;
- Entender a necessidade de suporte para correção de erros que possam aparecer depois da concepção do sistema.

# Documentação do produto

Documento com formato adequado ao perfil do público que utilizará o sistema ou produto. A linguagem deve se clara e os termos e construções devem estar de acordo com o nível cultural e técnico do usuário final.

Vejamos agora, alguns elementos que devem fazer parte dessa documentação e qual a sua finalidade.



## Manual de introdução

Descreve as funcionalidades do sistema, como o usuário pode utilizar, os pré-requisitos necessários para funcionar.

## Manual de referência

Descreve facilidades do uso do sistema, informa os erros que podem ocorrer e como agir quando encontrá-los.



## Documento de instalação

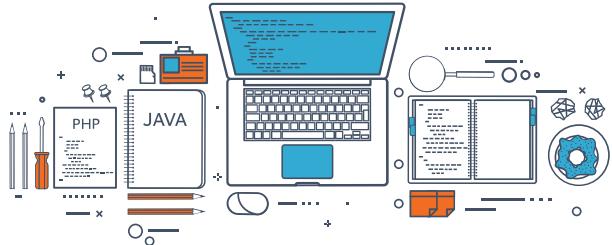
Descrição de como instalar o sistema, plataformas de operação, pré-requisitos necessários.

## Referência rápida

Documento com um resumo das funcionalidades, atalhos de procedimentos, principais funções utilizadas, e mensagens de erros mais comuns.

**Uma referência básica para o padrão manual é o [IEEE Std. 1063-2001](#) <<https://standards.ieee.org/findstds/standard/1063-2001.html>> .**





## Documentação do software

Processo que descreve as partes do código fonte, requisitos necessários, arquitetura do sistema.

**Essa documentação é bastante útil para o desenvolvedor no processo de melhoria ou correção do produto.**

## Manutenção do software

Consiste em corrigir defeitos ou deficiências encontradas pelos administradores ou usuários do produto. A manutenção também pode ser considerada um processo de melhoria das funcionalidades do software.



## Refatoração

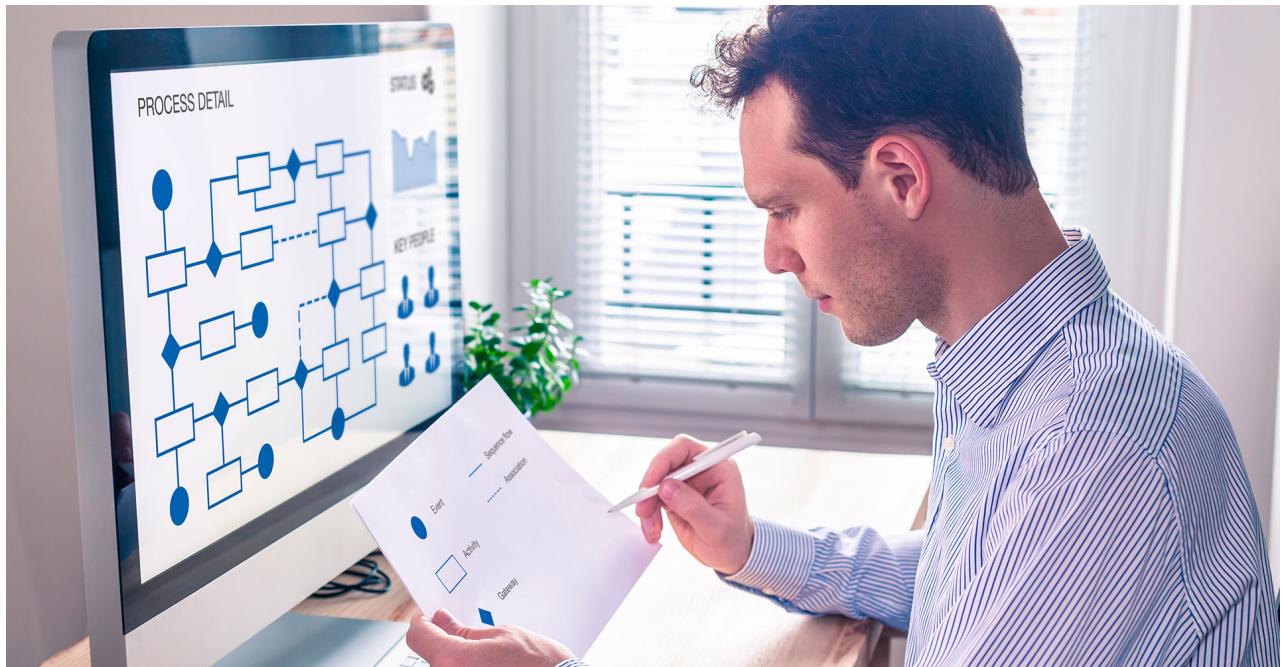
Dentro do processo de manutenção do software existe a refatoração, que tem como objetivo melhorar um sistema de software, modificando sua estrutura interna, sem alterar o comportamento interno.

# Separação Estática

Processo pelo qual identifica-se variáveis que apresentem algum tipo de não conformidade com o programa.

O processo leva a identificação do código onde a variável afeta sua funcionalidade.

## Documentação do processo



Documentação do processo. (Fonte: NicoELNino / Shutterstock)

## Cronogramas

Documentação utilizada por gerentes de projetos, executivos e gerentes funcionais, para acompanhar o andamento do projeto.



## Relatórios

Documentação de acompanhamento de recursos utilizados durante o andamento do projeto.



## Documentos técnicos

Descreve estratégias de como chegar ao resultado final, registram os erros, problemas e ideias que ocorrem durante o projeto, e as razões que foram utilizadas para as tomadas de decisões.



## Comunicação

Estabelece a forma de comunicação entre os membros do projeto.



## Padronização de processos

Estabelece o formato e a cadênciа de como o processo deve ser implementado. Essa padronização pode seguir o formato definido pela empresa, organização, ou um formato mais abrangente, como nacional ou internacional.



(Fonte: seamuss, DStarky / Shutterstock)

## Referências

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

O processo cascata.

## Explore Mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## Aula 8: O desenvolvimento do software em cascata



## Introdução

Nesta aula iremos demonstrar o modelo de desenvolvimento de software em cascata.

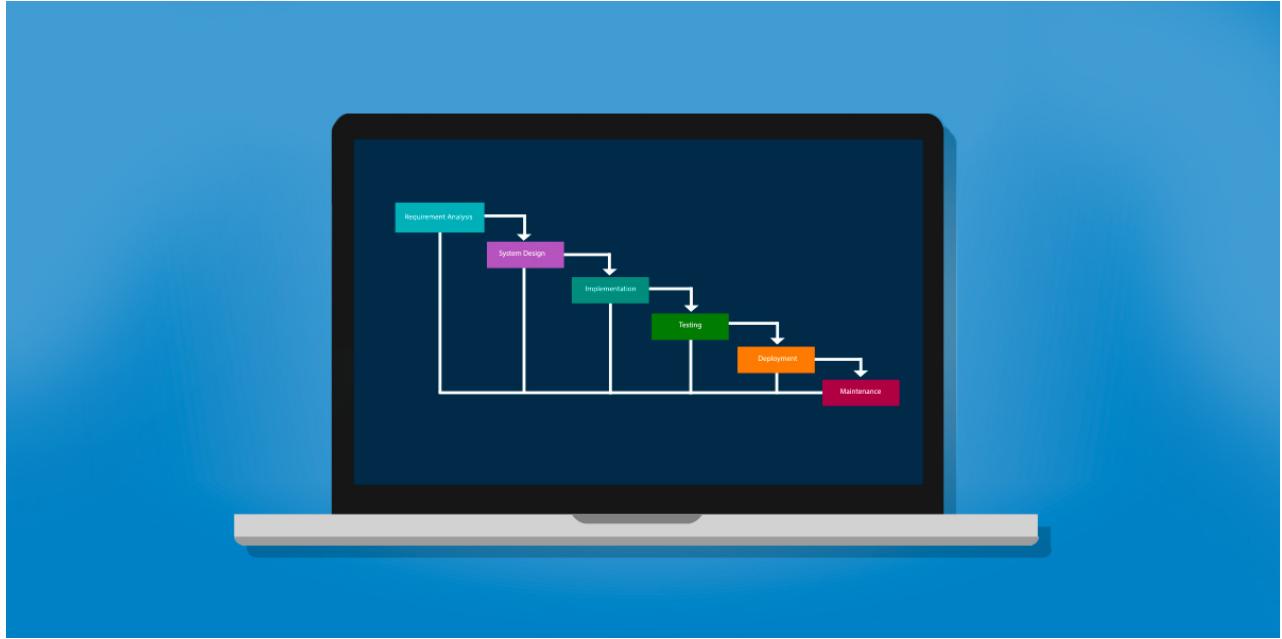
Inicialmente, não se seguia um modelo de desenvolvimento de software. Os desenvolvedores baseavam-se em suas próprias experiências e não havia uma forma definida e estruturada para o desenvolvimento. O resultado era softwares que entravam em produção com erros não testados e com a obrigatoriedade de correções após a fase de implementação.

O modelo em cascata, também conhecido como “water fall” ou “Top-Down” tem como característica utilizar as etapas, que foram estudadas anteriormente, de um modo sequencial e constantemente para frente.

## Objetivos

- Conhecer o processo em cascata, modelo de desenvolvimento de software sequencial, dentro do modelo de desenvolvimento de software;
- Entender as vantagens do modelo e suas limitações;
- Analisar as etapas iniciais do processo de desenvolvimento de software e aplicá-las no modelo em cascata.

# Modelo inicial



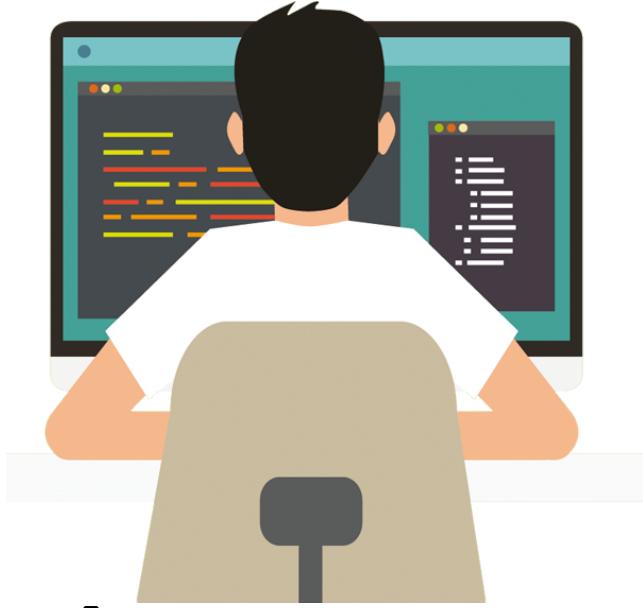
Modelo de desenvolvimento de software em cascata (Fonte: Shutterstock/Bakhtiar Zein).

# Modelo balbúrdia

O **Modelo balbúrdia** foi um modelo de desenvolvimento de software que existiu antes dos anos 70 e 80. Ele tinha uma maneira de trabalhar, na sua maioria, sem um padrão pré-estabelecido e fundamentava-se na própria experiência do programador, isso inclui o conhecimento tácito.

Esse modelo podia ser descrito por um ciclo de duas fases:

- 1. Correção**
- 2. Implementação**



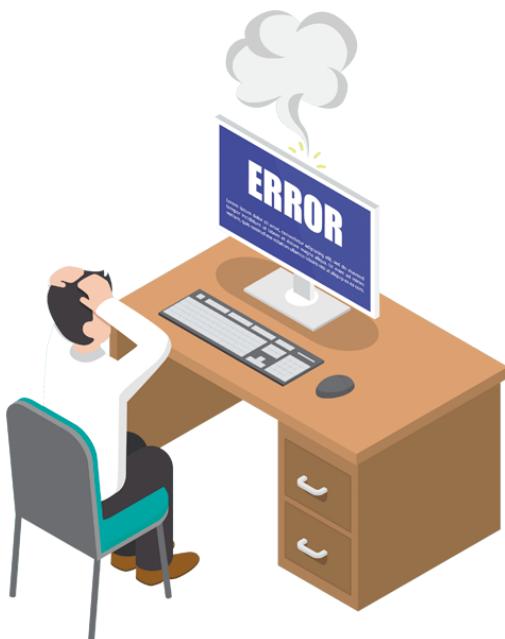
Desenvolvedor (Fonte: Shutterstock).

## Codifica - remenda

surge também nessa época uma pequena evolução do modelo Balburdia para **MODELO CODIFICA-REMENDA**. Esse modelo também não exige nenhuma sofisticação técnica ou gerencial, e não existe um compromisso do programador em fazer exatamente o que foi acordado.

Deste modo, a insatisfação e pressão dos usuários começam a crescer, e os usuários começam a duvidar da qualidade dos programas desenvolvidos.

Esses conflitos dentro da área de sistemas acarretaram na busca por novas soluções. Assim, já nas décadas de 70 e 80 há uma grande preocupação com a organização dos processos de desenvolvimento de software com o objetivo de aproximar o desenvolvimento do usuário e segmentar algumas atividades (formar especialistas).



📷 Erro na utilização do sistema. (Fonte: Shutterstock).

## Modelos de processos de desenvolvimento de software

### Ciclo da vida do projeto

Conjunto de atividades descritas e ordenadas que segue um fluxo contínuo de informações e relacionamentos para auxiliar o acompanhamento de um projeto.

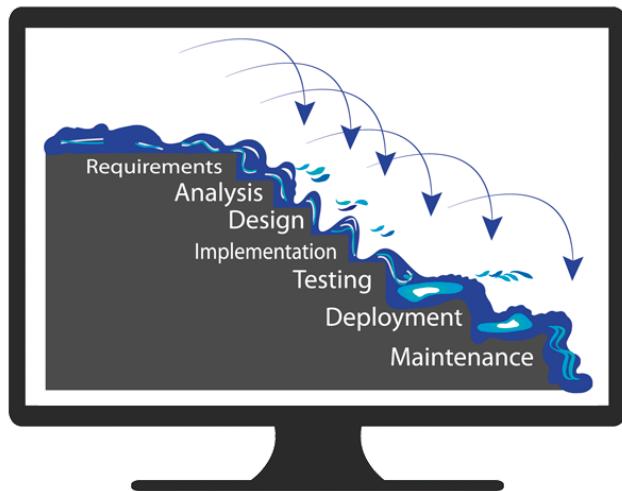


# Modelo de processo de cascata

Com a chegada dos anos 80 e com base nas práticas de engenharia tradicional começa a se desenhar um novo modelo com características mais rígidas e menos administrativo, quando comparado com os outros.

Com isso, dentro dessa evolução, surge o **MODELO CASCATA (CLÁSSICO)** ou também chamado de **CICLO CASCATA**.

O Modelo Cascata tem como característica o desenvolvimento de atividades sequenciais ordenadas. Assim, podemos dizer que esse modelo se propõe a criar um desenvolvimento alinhado, em ciclo de vida com fases sequenciais e continuas que possibilitem não só acompanhamento melhor do projeto, como também uma relação linear e “sempre para frente”, ou seja, para começar a fase 2 eu tenho que primeiro concluir a fase 1.



**Vejamos um exemplo para entender melhor.**

A etapa de **Projeto** só poderá ser iniciada após a finalização da etapa de requisitos.



Exemplo de gráfico em cascata.

## Vantagens do modelo cascata

- Permite ponto de controle bem definidos.
- Requer o mínimo de documentação.
- Simples de implementar.
- Possui fácil gestão.

## Desvantagens do modelo cascata

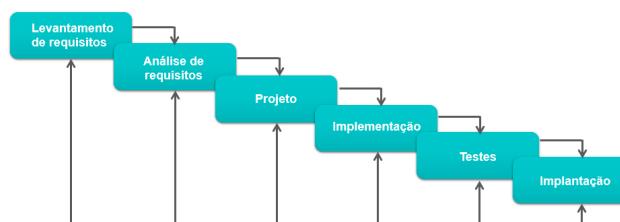
- O levantamento de requisitos ocorre somente na primeira fase.
- Não é possível corrigir erros nas fases anteriores.
- Não prevê manutenção.
- Uma etapa é realizada somente após a finalização do antecedente.
- Se ocorrer um atraso em uma fase todas as fases seguintes são afetadas (dependência).
- Só o gerente do projeto tem uma visão global do projeto.

## Modelo cascata com realimentação

### Modelo em cascata com realimentação

Modelo que permite a revisão de fases anteriores e a superposição entre as fases. Esse modelo é uma variante do modelo cascata tradicional que permite a realimentação, ou seja, correções durante outras fases do processo. Assim, a medida que os problemas vão sendo descobertos eles poderão ser corrigidos mesmo em fases anteriores.

**Vejamos o exemplo abaixo para entender melhor.**





## Vantagens do modelo cascata com realimentação

- Possibilidade de correção de erros de fases anteriores, ou seja, podemos correção a medida que os problemas vão sendo descobertos.
- Permite a superposição entre fases, ou seja, pode existir duas fases sendo realizadas ao mesmo tempo.
- Possibilita validação do usuário a cada fase.
- É adequado a projetos de pequena duração, tais como os projetos de desenvolvimento de software pessoal (PSP).

## Desvantagens do modelo cascata com realimentação

- Dependendo da quantidade de revisões e realimentações, o processo pode se tornar difícil de gerenciar.
- Retroceder às fases anteriores, principalmente nas fases de teste e implantação, pode aumentar o custo do projeto.

## Referências

---

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

---

- Processo iterativo.

## Explore Mais

---

Pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.

## Aula 9: O processo iterativo e incremental



## Apresentação

Nesta aula, iremos demonstrar o modelo de desenvolvimento de software iterativo.

Como vimos anteriormente, o modelo em cascata, também conhecido como “water fall” ou “Top-Down”, tem como característica utilizar as etapas que foram estudadas anteriormente de um modo sequencial e constantemente para frente, mas o processo em si possui algumas características, como:

- Passa para a fase subsequente somente quando a fase atual estiver completa.
- Não ser possível corrigir erros em fases já completas.
- O resultado do software somente será conhecido no final de todo o processo.

Para resolver algumas dessas características, foi criada uma variante do processo com retro alimentação, ou seja, a possibilidade de corrigir e voltar em etapas anteriores. No processo iterativo e incremental, essas ideias e correções são feitas em pequenas porções ao invés do processo como um todo. Essa abordagem intercala as atividades de especificação, desenvolvimento e validação.

O sistema é desenvolvido como uma série de versões (incrementos), de maneira que cada versão adiciona funcionalidade à anterior.

---

## Objetivos

- Conhecer o processo iterativo e incremental, modelo de desenvolvimento de *software* variante do processo em cascata;
- Entender as vantagens do modelo e suas limitações;
- Analisar as etapas iniciais do processo de desenvolvimento de software e aplicá-las no modelo iterativo.

# Introdução

No desenvolvimento de software, temos vários modelos. Na aula anterior, estudamos o mais utilizado, o modelo em cascata.

**Você alguma vez já ouviu falar sobre desenvolvimento iterativo e incremental?**

Esse tipo de desenvolvimento, remete as metodologias ágeis. Vamos entender melhor.

## Modelo Iterativo

O modelo se baseia na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar o desenvolvimento através de várias versões até que um sistema adequado seja desenvolvido.

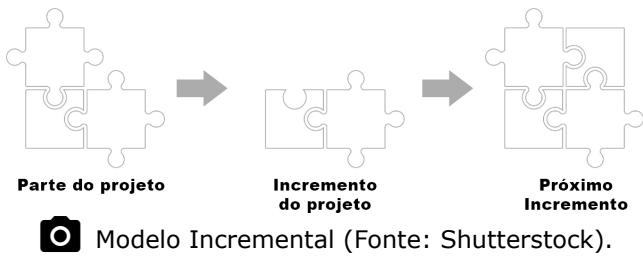
Caracteriza-se pela seleção de uma parte do projeto onde o grupo de desenvolvedores identifica, especifica, implementa e testa a iteração. Se esta atender às especificações, a equipe passa para a próxima iteração.



Modelo Iterativo (Fonte: Shutterstock).

## Modelo Incremental

Modelo que se baseia na ideia de aumento do âmbito do sistema, ou seja, na criação de novas versões para o modelo proposto.



# Modelo Iterativo e Incremental

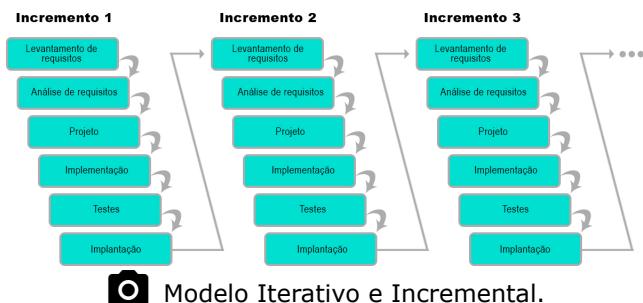
Metodologia de desenvolvimento de software que define um subconjunto de requisitos e utiliza o modelo em cascata para sua realização.

Cada porção do ciclo ou incremento incorpora alguma funcionalidade necessária para o cliente. Frequentemente, os incrementos iniciais incluem a funcionalidade mais importante ou mais urgente. Desta forma o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado. Caso contrário, só o incremento que estiver em desenvolvimento no momento precisará ser alterado e, uma nova funcionalidade será definida para incrementos posteriores.

Vantagens na utilização do modelo:

- O custo de acomodar as mudanças nos requisitos do cliente é reduzido.
- É mais fácil obter feedback dos clientes sobre o desenvolvimento que foi feito.
- É possível obter entrega e implementação rápida de um software útil ao cliente, mesmo se toda a funcionalidade não for incluída.

## Exemplo

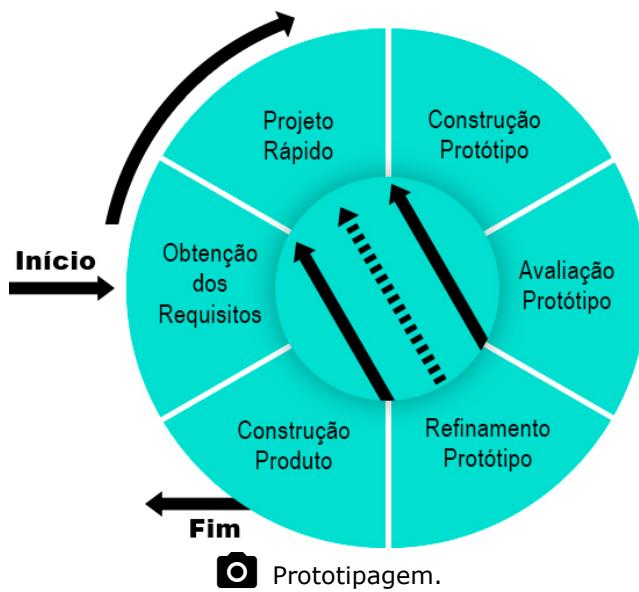


# Modelo de Prototipagem

Um protótipo é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções.

Enquanto o sistema está em projeto, um protótipo do sistema pode ser usado para a realização de experimentos de projeto visando à verificação da viabilidade da proposta.

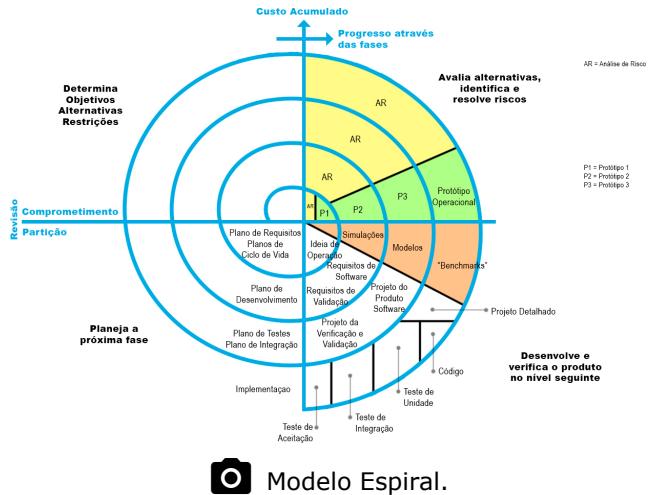
O desenvolvimento rápido e iterativo do protótipo é essencial para que os custos sejam controlados e os stakeholders do sistema possam experimentá-lo no início do processo de software.



## Modelo Espiral

O Modelo espiral se assemelha com o prototipagem, mas inclui um fator: a análise de risco. Funciona de forma iterativa, incremental, mas com uma etapa onde pode ser tomada a decisão de se interromper ou não o processo.

### Exemplo



Modelo Espiral.

Neste modelo o processo de software é representado como uma espiral. Cada volta na espiral representa uma fase do processo de software: a volta mais interna preocupar-se com a viabilidade do sistema; o ciclo seguinte, com a definição de requisitos; o seguinte, com o projeto do sistema, e assim por diante.

Cada volta é dividida em quatro setores:

- Definição de objetivos
- Avaliação e redução de riscos
- Desenvolvimento e validação

## Referências

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Próximos Passos

- Processo unificado.

## Explore Mais

Modelo em Iterativo e incremental:

[http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/21)  
[/pages/21](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/21)  
<<http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/21>>

Modelo em espiral:

[http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/33)  
[/pages/33](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/33)  
<<http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/33>>

Prototipação:

[http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/31)  
[/pages/31](http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/31)  
<<http://estacio.bv3.digitalpages.com.br/users/publications/9788579361081/pages/31>>

## Aula 10: O processo iterativo e incremental



## Apresentação

Nesta aula, iremos demonstrar outros modelos de desenvolvimento de software.

Como vimos anteriormente, os modelos seguiam um padrão de organização onde se criavam etapas ou módulos caracterizados pelas ações que se tomariam durante um período determinado. Esses períodos se caracterizavam por fases que, após serem finalizadas, davam inicio a uma outra fase, com características diferentes.

Cada fase representava uma ação que possuía um inicio e um fim, mesmo nos casos de realimentação.

Para os modelos aqui apresentados, não estão somente levados em conta o planejamento e ação, mas também os valores e os recursos envolvidos.

---

## Objetivos

- Conhecer outros processos utilizados no processo de desenvolvimentos de *software*;
- Entender as vantagens dos modelos e suas limitações;
- Analisar as etapas iniciais do processo de desenvolvimento de *software* e compará-los.

# Processo de desenvolvimento Ágil

## Método Ágil

É um conjunto de diretrizes e metodologias que cria uma estrutura conceitual para desenvolver projetos de desenvolvimento de software.

Baseado em um manifesto criado por programadores veteranos que já tinham passado por inúmeras experiências diferentes no campo de desenvolvimento de software, o **Manifesto Ágil** [<http://agilemanifesto.org/iso/ptbr/manifesto.html>](http://agilemanifesto.org/iso/ptbr/manifesto.html) tem como foco as pessoas e não as ferramentas.



Fonte: Shutterstock/Llin Sergey.

## Método XP

Também conhecido como eXtreme Programming, é um método que pertence à metodologia ágil de desenvolvimento de software.

**O modelo propõem uma série de práticas focados em pessoas, ou seja, na equipe de desenvolvimento.**

É baseado em 5 valores:

## Comunicação

2

## Coragem

3

## Feedback

4

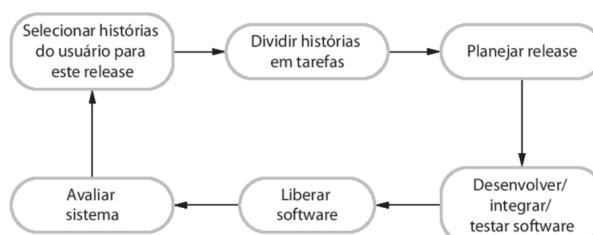
## Respeito

5

## Simplicidade

# Ciclo de um release em XP

Como apresentado por Sommerville, “em XP, os requisitos são expressos em formas de cenários (chamados de histórias do usuário), que são implementados diretamente como uma série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código. Quando o novo código é integrado ao sistema, todos os testes devem ser executados com sucesso.”



Vejamos algumas práticas do método XP:



## **Reuniões em pé**

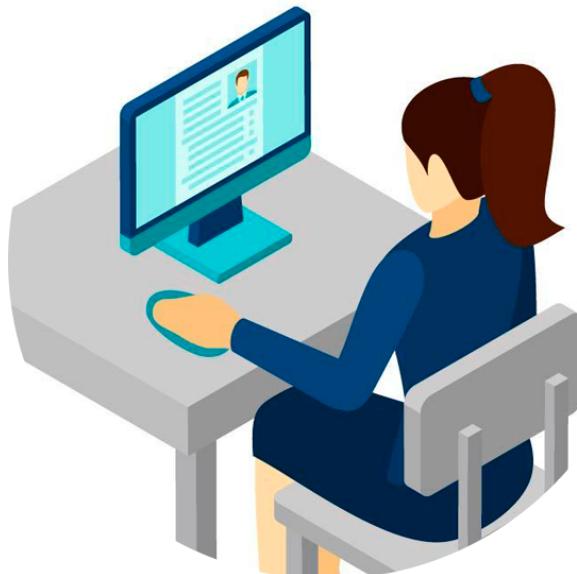
Utilizadas para não perder o foco no assunto.



## **Programações em par**

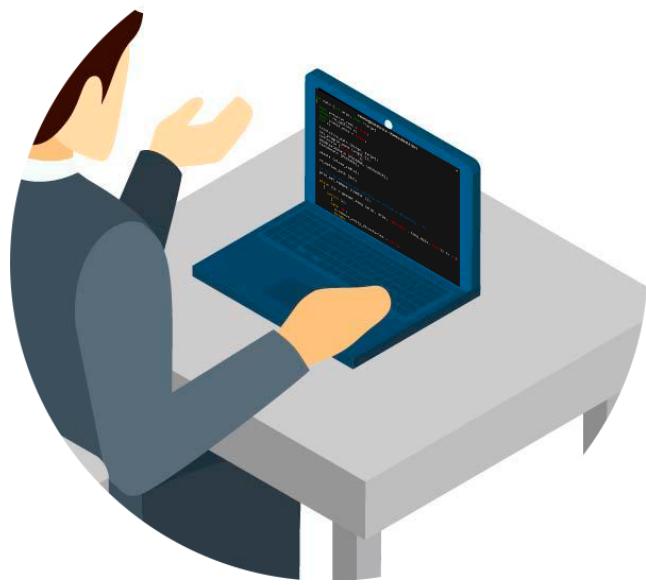
Todo o código é implementado por uma dupla, usando o mesmo computador. Ambos com o objetivo de resolver o mesmo problema.

Um terá o papel do condutor, digitando o código, e o outro o papel de navegador, com o objetivo de revisar o código de forma a evitar erros de programação ou sugerir melhores estratégicas de implementação.



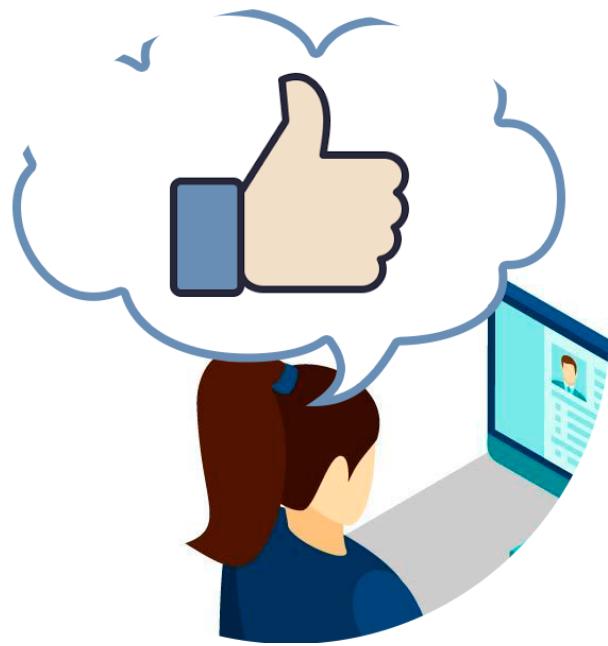
## **Desenvolvimento guiado por testes**

Desenvolvimento guiado por testes: Trabalhar com teste automatizado de forma acompanhar se as funcionalidades implementadas atendem ao requisito e ao cliente. Dentre os testes podemos trabalhar com os testes de unidade e os testes de aceitação.



## **Posse coletiva**

O código fonte não pertence a ninguém, é de todos e todos podem utilizá-lo sem necessidade de permissão.



## **Pequenas versões**

Pequenas versões aceitas pelo cliente ajudam na aceitação do programa completo.



## **Ritmo sustentável**

Utilizar o tempo de trabalho dentro do especificado. Sem horas adicionais. ( 40 horas por semana ).



## **Padrão de codificação**

Após a formação da equipe, deverá ser estabelecida algumas regras, como o padrão de desenvolvimento a ser usado. A escolha de um padrão visa um rápido entendimento dos códigos uns dos outros.

## **Método Scrum**

Metodologia que tem como filosofia o Manifesto Ágil.

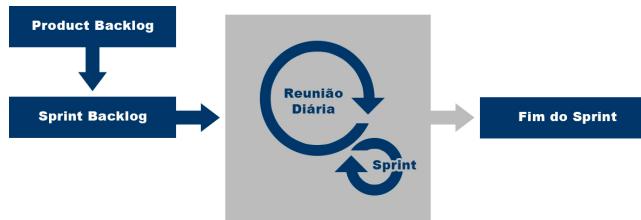
Possui papel bem definido para as atividades durante todo o processo. Uma vez levantadas as questões a serem trabalhadas, é determinado um período de tempo para a realização de um determinado requisito.

Durante esse intervalo, são feitas reuniões diárias para acompanhamento do andamento das atividades.

Os Três Pilares Fundamentais do Scrum

- Transparência: Transparência dos processos, requisitos de entrega e status.
- Inspeção: Inspeção constante de tudo o que está sendo realizado.

- Adaptação: Adaptação, tanto do processo, quanto do produto às mudanças.



# Fundamentos Básicos do Scrum

## 1. Papéis

1.1- Dono do Produto (Product Owner): É o responsável por gerenciar o Backlog do Produto e garantir o valor do trabalho executado pela Equipe de Desenvolvimento.

1.2 - Scrum Master: É a pessoa que mais conhece o Scrum dentre todos os papéis. Ele tem o papel de facilitador, ajudando a equipe a resolver problemas, realizando melhorias no uso do Scrum, além de manter a equipe focada em suas tarefas. Vale ressaltar que o Scrum Master não é um gerente.

1.3 – Time de Desenvolvimento (Team Scrum): É a equipe (time) responsável pelo desenvolvimento do projeto. Os membros da equipe deve ser multidisciplinar e multifuncional, possuindo todo conhecimento necessário para a criar um incremento no trabalho. Geralmente o time de desenvolvimento é constituído por três a nove pessoas. Tanto o Scrum Master o Dono do Produto não estão incluídos no tamanho do Time de desenvolvimento.

## 2. Artefatos

2.1 – Backlog do Produto: É um documento que contém todos os itens que devem ser desenvolvidos durante o projeto. Sendo o Dono do produto responsável por criar e manter este documento. É importante ressaltar que os itens do Backlog do Produto devem ser priorizados em função do Retorno do Investimento (ROI), onde os itens que apresentam maior valor para o negócio devem ser desenvolvidos primeiro.

2.2 – Backlog da Sprint: É um conjunto de itens selecionados para serem implementados durante a Sprint. Se durante o desenvolvimento da Sprint, for identificada a necessidade de novas tarefas ou a remoção de alguma tarefa,

somente os membros do Time de Desenvolvimento podem realizar essa alteração no Backlog da Sprint.

2.3 – Incremento do Produto: Ao término de cada Sprint, o Time de Desenvolvimento entrega um incremento do produto, o qual é o resultado do que foi produzido durante a Sprint. Ao final de uma Sprint, uma funcionalidade só é considerada pronta depois de passar por todas as etapas determinadas pelo Time de Desenvolvimento. Caso a funcionalidade não tenha sido concluída, a mesma deve retornar ao Backlog do produto para que seja incluída em uma próxima sprint.

### **3. Cerimônias**

3.1 – Sprint: No Scrum, o trabalho é desenvolvido através de interações, denominadas Sprints. Cada Sprint tem duração de 2 a 4 semanas.

3.2 - Reunião de Planejamento da Sprint: Ocorre sempre no início de cada Sprint, com o objetivo de planejar o que será feito na Sprint.

3.3 - Reunião Diária: É uma reunião simples e importante, na qual cada membro do Time de Desenvolvimento deve responder sobre o que já fez, sobre o que pretende fazer e se há algum impedimento para a conclusão da tarefa sob sua responsabilidade. Geralmente essa reunião dura no máximo 15 minutos.

3.4 - Revisão da Sprint: Tem como objetivos apresentar o que a equipe fez durante a Sprint e entregar o produto ao Dono do Produto (Product Owner). Cabe ao Dono do Produto aceitar ou rejeitar a Sprint com base no que foi apresentado.

3.5 - Retrospectiva da Sprint: É uma reunião que tem como foco o aprimoramento do processo, analisando o que houve de bom e o que pode ser melhorado em uma Sprint. Todos os membros da equipe Scrum participa desta reunião.

## **Processo unificado**

### **Rup**

Também conhecido como Rational Unified Process, é um processo que faz parte da engenharia de software.

Ele é baseado em disciplinas em que cada uma distribui tarefas e responsabilidades para os envolvidos no desenvolvimento do software.

## **Essas disciplinas são semelhantes às que estudamos anteriormente:**

- Modelagem de negócios;
- Implementação;
- Requisitos;
- Teste;
- Análise e Design;
- Implantação.

Ainda no RUP, existem 3 disciplinas que servem de suporte e apoio ao ambiente:

### **Configuração e mudanças**

Acompanham mudanças, configurações e status/medidas onde são armazenados e que servirão de base para o andamento do projeto.

### **Projeto**

Abrange questões como gestão de pessoas, orçamento, contratos.

### **Ambiente**

Atividades que dão suporte à equipe de desenvolvimento, como os itens de IT, servidores, ferramentas.

---

**Essas disciplinas têm suas responsabilidades e funções variadas, dependendo da fase que se encontra o projeto.**

---

No processo RUP, o tempo está dividido em 4 fases:

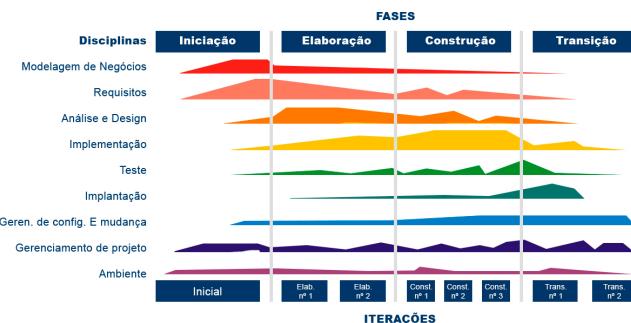


Imagen do software RUP, com as duas dimensões e as fases incluídas.

## Referências

GUSTAFSON, Davis A. **Engenharia de software**. 8. ed. São Paulo: Pearson Education, 2007. cap. 8 e 13.

PAULA FILHO, Wilson de. **Engenharia de software**: fundamentos, métodos e padrões. 3. ed. São Paulo: LTC, 2009. cap. 1, 5 e 21.

SOMMERVILLE, Ian. **Engenharia de software**. 1. ed. Porto Alegre: Artmed, 2003. cap. 10.

## Explore Mais

XP: [<http://www.extremeprogramming.org>](http://www.extremeprogramming.org)

Scrum: [<https://www.scrum.org/resources/what-is-scrum>](https://www.scrum.org/resources/what-is-scrum)