

# TECNOLOGIA DA INFORMAÇÃO

Engenharia de Software



Livro Eletrônico



# SUMÁRIO

Apresentação .....	4
Engenharia de Software .....	5
1. Visão Geral da Engenharia de Software .....	5
2. Ciclo de Vida do Software .....	6
3. Fluxo de Processo .....	7
3.1. Fluxo de Processo Linear .....	7
3.2. Fluxo de Processo Iterativo .....	8
3.3. Fluxo de Processo Evolucionário .....	8
3.4. Fluxo de Processo Paralelo .....	8
4. Processo de Software .....	9
4.1. Conceitos Principais Relacionados ao Tema .....	9
4.2. Visão Geral Sobre o Processo de Desenvolvimento de Software .....	30
5. Modelos de Gestão: Catedral e Bazar .....	37
5.1. Catedral (Tradicional) .....	37
5.2. Bazar (Colaborativo) .....	37
6. Outros Conceitos Importantes .....	38
6.1. Abstração .....	38
6.2. Modelagem .....	40
7. Conceitos Principais Relacionados à Análise Estruturada .....	40
8. O Paradigma da Orientação a Objetos .....	42
9. Aspectos da Orientação a Objetos .....	42
9.1. Classificação .....	42
9.2. Identidade .....	43
9.3. Ênfase na Estrutura de Objetos .....	43
9.4. Compartilhamento .....	43
9.5. Polimorfismo .....	43
9.6. Encapsulamento .....	43

10. Modelagem Orientada a Objetos .....	44
11. Análise Orientada a Objetos x Análise Estruturada .....	44
Resumo.....	45
Questões Comentadas em Aula .....	47
Questões de Concurso .....	49
Gabarito.....	81
Referências.....	82

## APRESENTAÇÃO

Olá, querido(a) amigo(a), tudo bem?



Quer conquistar a sua vitória? Então, tenha uma postura grandiosa! Todos que venceram grandes desafios se alimentavam com pensamentos **positivos, engrandecedores, de esperança e de coragem**.

Mesmo diante das adversidades, conserve a elegância e uma autoimagem positiva. Essa autoimagem é como se fosse um combustível que impulsionará as suas ações. #FicadaDica #OSegredo

Rumo então à aula sobre **Engenharia de Software, com foco no ciclo de vida do software e nas metodologias de desenvolvimento de software**.

Em caso de dúvidas, acesse o fórum do curso ou entre em contato.

Um forte abraço,

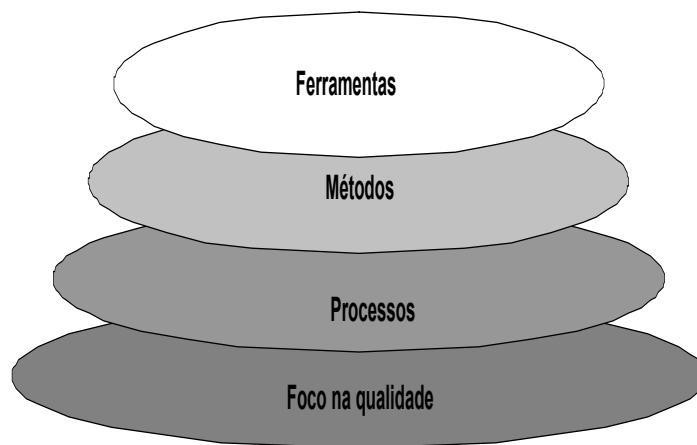


O conteúdo deste livro eletrônico é licenciado para 61984693488 Martins Rodrigues - 00193743132, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

# ENGENHARIA DE SOFTWARE

## 1. VISÃO GERAL DA ENGENHARIA DE SOFTWARE

- é uma área da computação que visa abordar de modo sistemático as **questões técnicas e não técnicas** no **projeto, implantação, operação e manutenção** no desenvolvimento de um software;
- é uma disciplina que se **ocupa de todos os aspectos da produção de software**, desde **os estágios iniciais de especificação do sistema até a manutenção desse sistema**, depois que ele entrou em operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de software com alta qualidade;
- tem como **foco principal** estabelecer uma **abordagem sistemática de desenvolvimento**, através de ferramentas e técnicas apropriadas, dependendo do problema a ser abordado, **considerando restrições e recursos disponíveis**.
- A IEEE define **engenharia de software** como a aplicação de uma abordagem sistemática, disciplinada e quantificável de **desenvolvimento, operação e manutenção de software**.
- Já Friedrich Bauer conceitua como a **criação e a utilização de sólidos princípios de engenharia** a fim de obter software de maneira econômica, que seja confiável e que trabalhe em máquinas reais.
- De acordo com Pressman, a **engenharia de software é baseada em camadas, com foco na qualidade**. Essas camadas são: **ferramentas, métodos e processo**.



- Os **princípios de engenharia** de software definem a necessidade de formalidades para reduzir inconsistências e a decomposição para lidar com a complexidade.

## 2. CICLO DE VIDA DO SOFTWARE

- O ciclo de vida de um software, entre outras características, está relacionado aos estágios de **concepção, projeto, criação e implementação** (CESPE/Unb-2013).
- Nesse contexto, **Sommerville destaca as quatro atividades fundamentais de processos de software**, comuns a todos eles, que são:

Especificação de Software	São definidas as funcionalidades do software e restrições para sua operação.
<b>Projeto e Implementação de Software</b>	O software que atenda à especificação deve ser <b>produzido</b> .
<b>Validação de Software</b>	O software deve ser <b>avaliado</b> para garantir que ele faça o que o cliente deseja.
<b>Evolução do Software</b>	O software <b>evolui</b> para atender às necessidades de mudança do cliente.

Segundo o autor, essas atividades são organizadas de modo diferente nos diversos processos de desenvolvimento. Como exemplo, no modelo em cascata são organizadas em sequência, ao passo que, no desenvolvimento evolucionário, elas são intercaladas. **Como essas atividades serão organizadas dependerá do tipo de software, pessoas e estruturas organizacionais envolvidas.**

- Pressman (2011) destaca que **uma metodologia de processo genérica para Engenharia de Software compreende cinco atividades**:

<b>Comunicação</b>	Antes de iniciar o trabalho, é de vital importância comunicar-se e colaborar com o cliente (e outros interessados, como: executivos, usuários finais, engenheiros de software, o pessoal de suporte etc.). A intenção aqui é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.
<b>Planejamento</b>	Estabelecimento do <b>plano de projeto de software</b> , que descreve as <b>tarefas</b> técnicas a ser conduzidas, os <b>riscos</b> prováveis, os <b>recursos</b> que serão necessários, os <b>produtos</b> resultantes a ser produzidos e um <b>cronograma</b> de trabalho.
<b>Modelagem</b>	Criação de <b>modelos representativos do software</b> (para melhor entender as necessidades do software e o projeto que irá atender a essas necessidades).

## Construção

Essa atividade combina **geração de código** (manual ou automatizada) e **testes** necessários para revelar erros na codificação.

### Entrega (Implantação)

**Entrega do produto ao cliente**, que irá avaliá-lo e fornecer *feedback* baseado na avaliação.

- As atividades metodológicas são **complementadas** pelas atividades de apoio, como: controle e acompanhamento do projeto, administração de riscos, garantia da qualidade, gerenciamento da configuração de software, dentre outras.
- As sequências de atividades trazidas pelo Pressman e Sommerville são essencialmente iguais e estão falando do mesmo processo: o desenvolvimento de um software.

Deve-se notar ainda um aspecto do processo de software importante nesse contexto, chamado **fluxo de processo**, destacado a seguir.

## 3. FLUXO DE PROCESSO

O **fluxo de processo** descreve **COMO** são organizadas as atividades metodológicas, bem como as **ações** e **tarefas** que ocorrem dentro de cada atividade em relação à sequência e ao tempo, como ilustrado nas figuras desta seção.

### 3.1. FLUXO DE PROCESSO LINEAR

**Executa cada uma das cinco atividades metodológicas em sequência**, começando com a de comunicação e terminando com a entrega (ou emprego, ou implantação).

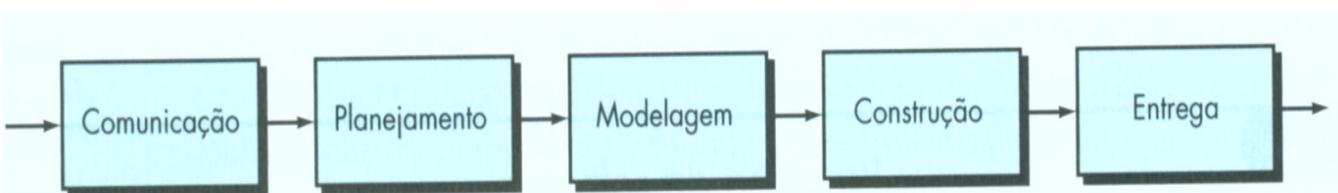


Figura. Fluxo de Processo Linear. Fonte: Pressman (2011)

### 3.2. FLUXO DE PROCESSO ITERATIVO

Repete uma ou mais das atividades antes de prosseguir para a seguinte, conforme listado na próxima figura.

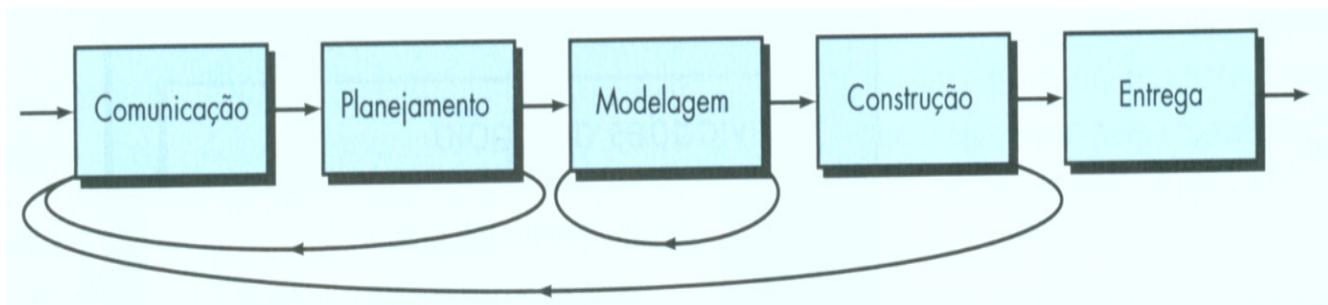


Figura. Fluxo de Processo Iterativo. Fonte: Pressman (2011)

### 3.3. FLUXO DE PROCESSO EVOLUÇÃOARIO

Executa as atividades de forma “circular”. Cada volta pelas cinco atividades conduz a uma versão mais completa do software.

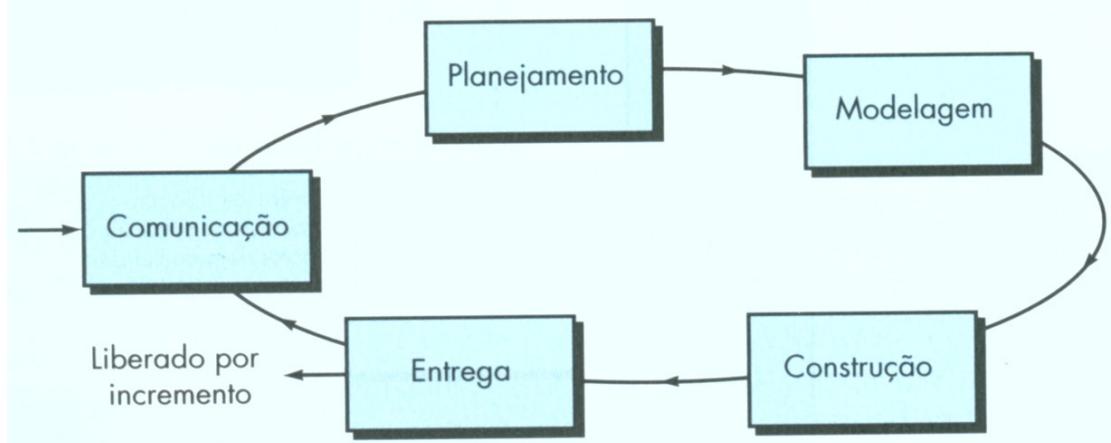
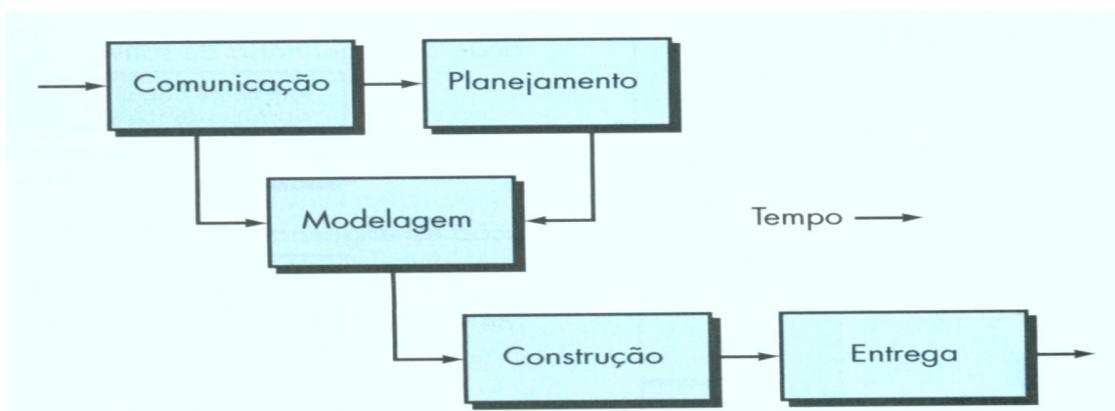


Figura. Fluxo de Processo Evolucionário. Fonte: Pressman (2011)

### 3.4. FLUXO DE PROCESSO PARALELO

Executa uma ou mais atividades em paralelo com outras atividades (como por exemplo a modelagem para um aspecto do software poderia ser executada em paralelo com a construção de um outro aspecto do software).



**Figura. Fluxo de Processo Paralelo. Fonte:** Pressman (2011)

## 4. PROCESSO DE SOFTWARE

### 4.1. CONCEITOS PRINCIPAIS RELACIONADOS AO TEMA

- **Software:** são os programas de computador e a documentação associada.
- **Processo de software:** é um conjunto de atividades, cuja meta é o desenvolvimento ou a evolução do software.

No **processo de software** um pequeno número de atividades de arcabouço aplicáveis ao desenvolvimento de qualquer software são definidas, independente do seu tamanho e complexidade.

Todos os modelos de processo de software, segundo Pressman (2011) podem acomodar as atividades metodológicas genéricas, aqui apresentadas, porém, **cada um deles dá uma ênfase diferente a essas atividades e define um fluxo de processo que invoca cada atividade metodológica de forma diversa.**

Há inúmeros processos de desenvolvimento propostos. Bezerra (2007) destaca que é um consenso na comunidade de desenvolvimento de software o fato de que não existe o melhor processo de desenvolvimento, aquele que melhor se aplica a todas as situações de desenvolvimento. Segundo o autor, cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento.

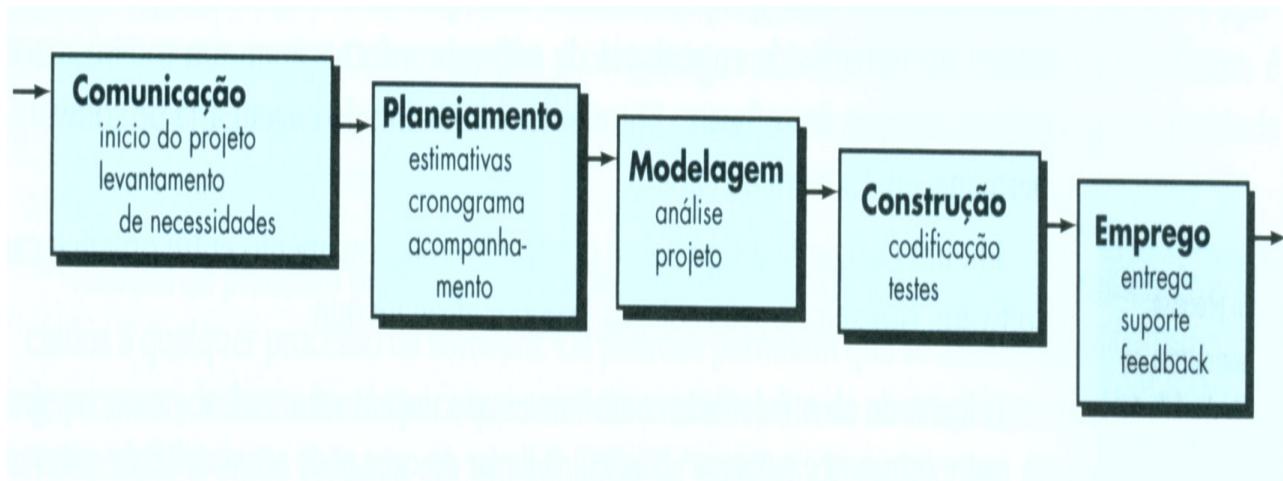
Vamos então ao estudo dos principais modelos de processo de software, importantes para a prova.

### Modelo em Cascata

Também chamado de clássico, ou linear, é o mais tradicional processo de desenvolvimento de software.

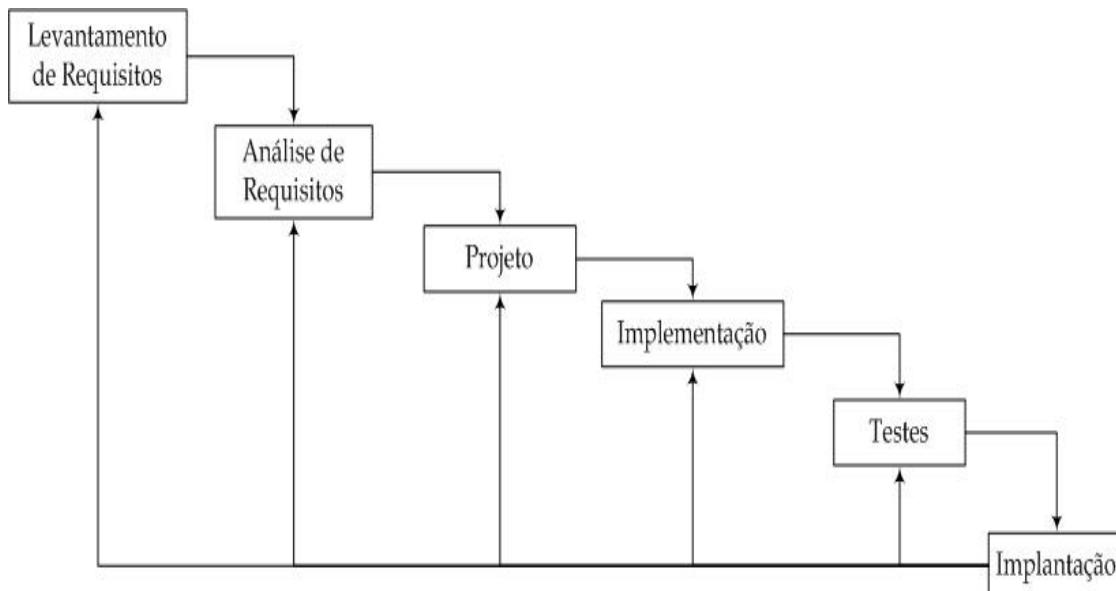
Esse modelo sugere uma abordagem **sequencial** e sistemática para o desenvolvimento de software, aplicando as atividades de maneira linear. Em cada fase desenvolvem-se artefatos (produtos de software) que servem de base para as fases seguintes.

Vide figura proposta por Pressman (2011) para esse modelo.

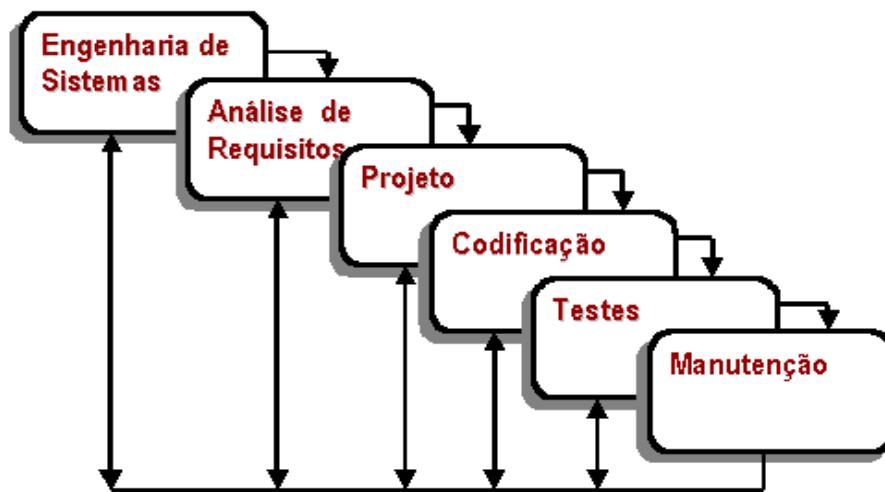


**Figura. Modelo Cascata, segundo Pressman (2011)**

Segundo proposta de Eduardo Bezerra, o **modelo do ciclo de vida clássico da engenharia de software** é dividido em **seis** atividades. São elas:



*Em seguida, a proposta de Kruchten (2003).*



O modelo em Cascata possui como **vantagem** principal a simplicidade para a sua aplicação e gerência.

No entanto, algumas **desvantagens** podem ser observadas:

- projetos reais raramente seguem este fluxo sequencial.
- dificuldade do cliente em declarar todas as suas necessidades no início do projeto.
- demora em apresentar resultados ao cliente.

Uma variação do modelo cascata, destacada por Pressman (2011) é o **modelo V**. Segundo o autor, à medida que a equipe de software desce em direção ao lado esquerdo do V, os requisitos básicos do problema são refinados em representações progressivamente cada vez mais detalhadas e técnicas do problema e de sua solução. Uma vez que o código tenha sido gerado, a equipe sobe o V, realizando uma série de testes que validem cada um dos modelos criados do outro lado do V.

Na realidade, segundo Pressman (2011) não existe diferença entre os modelos. O modelo V fornece uma maneira para visualizar como a verificação e as ações de validação são aplicadas ao trabalho de engenharia do ciclo de vida em Cascata.

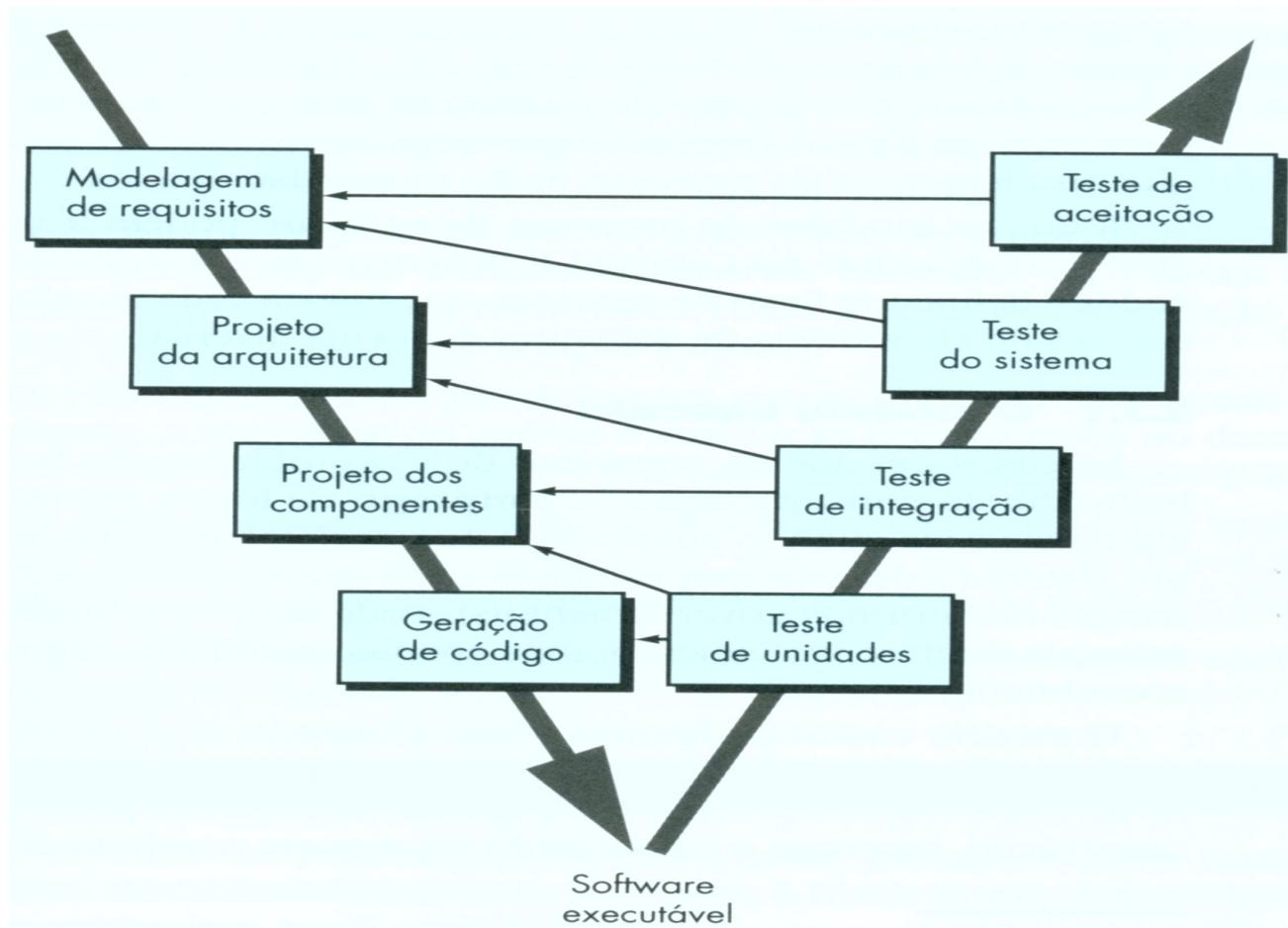


Figura. Modelo V. Fonte: Pressman (2011)

## DIRETO DO CONCURSO

**001.** (CESPE/2015/MEC/ANALISTA DE SISTEMAS) Julgue o item a seguir, a respeito da engenharia de software.

Na fase de engenharia de requisitos do software, do paradigma do ciclo de vida clássico da engenharia de software chamado de modelo cascata, são identificadas as necessidades do sistema do ponto de vista do desenvolvedor, sem a presença do solicitante.



Na fase de **engenharia de requisitos** do software, do modelo Cascata, são identificadas as necessidades do **ponto de vista do cliente** e **não do desenvolvedor**.

**Errado.**

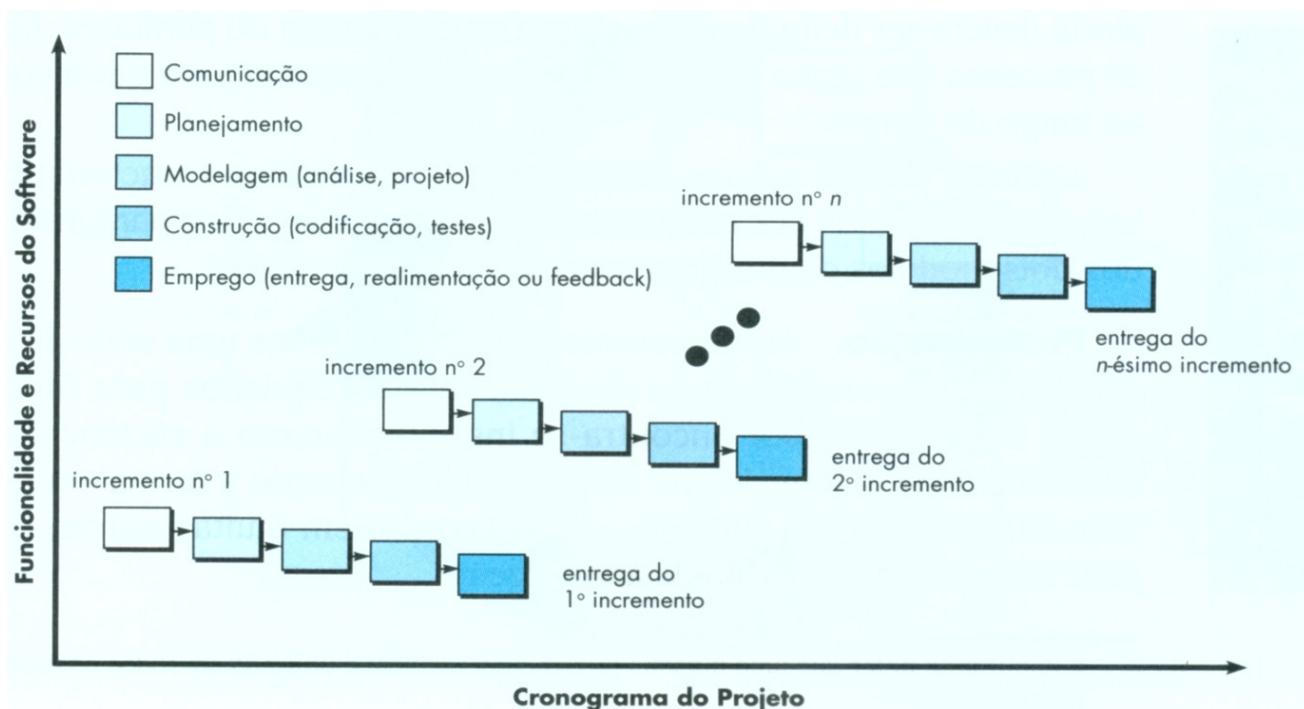
## Modelo Incremental

Este modelo foi proposto como uma alternativa ao modelo em cascata, aplicando-o **iterativamente**, tendo como objetivo a elaboração de um produto operacional a cada **incremento**. Os primeiros incrementos são versões simplificadas do produto final, mas oferecem capacidades que servem ao usuário, além de servir como uma plataforma de avaliação.

Em cada iteração uma parte é concebida como a menor unidade que pode ser implementada e ainda assim fornecer alguma funcionalidade útil para os usuários.

**Obs.:** Os modelos incrementais combinam elementos dos fluxos de processos lineares e paralelos.

Na figura seguinte, o modelo incremental aplica sequências lineares, de forma escalonada, à medida que o tempo vai avançando. **Cada sequência linear gera “incrementais” (entregáveis/ aprovados/liberados) do software.**



**Figura. Modelo Incremental. Fonte: Pressman (2011)**

Esse modelo possui como **vantagem** o fato de apresentar constantemente novas versões aos usuários. Pode ser aplicado também quando não houver mão-de-obra disponível para uma implementação completa, ou quando for necessário gerenciar riscos técnicos.

A figura seguinte é bem mais interessante para ilustrar a diferença entre incremental e iterativo!

- ▶ Incremental: são adicionados “pedaços completos”



- ▶ Iterativo: esboços ou pedaços incompletos do produto são adicionados a cada iteração



**Obs.:** Conforme destaca Pleeger (2004), para entender as diferenças entre o desenvolvimento incremental e iterativo, pense em um pacote de processamento de textos. Suponha que o pacote deva oferecer três tipos de recursos: criar textos, organizar textos (ou seja, recortar e colar) e formatar textos (como utilizar diferentes tamanhos e estilos de letras e números).

Para construir esse sistema utilizando o desenvolvimento incremental, poderíamos fornecer, na Versão 1, somente a função de criação de textos. Na Versão 2, poderíamos fornecer as funções de criação e organização de textos. Finalmente, na Versão 3, ofereceríamos as três funções.

De outra maneira, utilizando o desenvolvimento iterativo, forneceríamos formas primitivas das três funções logo na Versão 1. Por exemplo, podemos criar textos e, então, recortar e colar partes deles, mas as funções de recortar e colar podem ser ainda imperfeitas e lentas. Assim, na Versão 2, vamos dispor da mesma funcionalidade, porém teremos aprimorado a qualidade; agora as funções de recortar e colar são simples e rápidas. Cada versão aprimora a anterior de algum modo.

## DIRETO DO CONCURSO

**002.** (CESPE/2016/FUNPRESP/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO) O modelo de execução de projetos em cascata é caracterizado por fases que se entrelaçam e se sobrepõem. A abordagem incremental, por sua vez, assemelha-se ao planejamento em ondas sucessivas.



O item está **incorreto** em sua primeira parte.

Na **abordagem de execução de projetos em cascata**, todas as etapas são seguidas de forma **sequencial**, dessa forma as fases **não** se sobrepõem, tampouco se entrelaçam, como afirma o item. No modelo em cascata nós só avançamos para a próxima fase quando a anterior é concluída, em outras palavras, o **estágio seguinte não deve ser iniciado até que a fase anterior seja concluída**. Logo, voltar algumas etapas, dar saltos para frente ou sobrepor atividades não é permitido.

O **desenvolvimento incremental** é baseado na ideia de **desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões** até que um sistema adequado seja desenvolvido. Assim, o **modelo incremental** é simples: **à medida que o projeto vai evoluindo, e as incertezas diminuindo, planeja-se a próxima etapa ou onda, e assim por diante**.

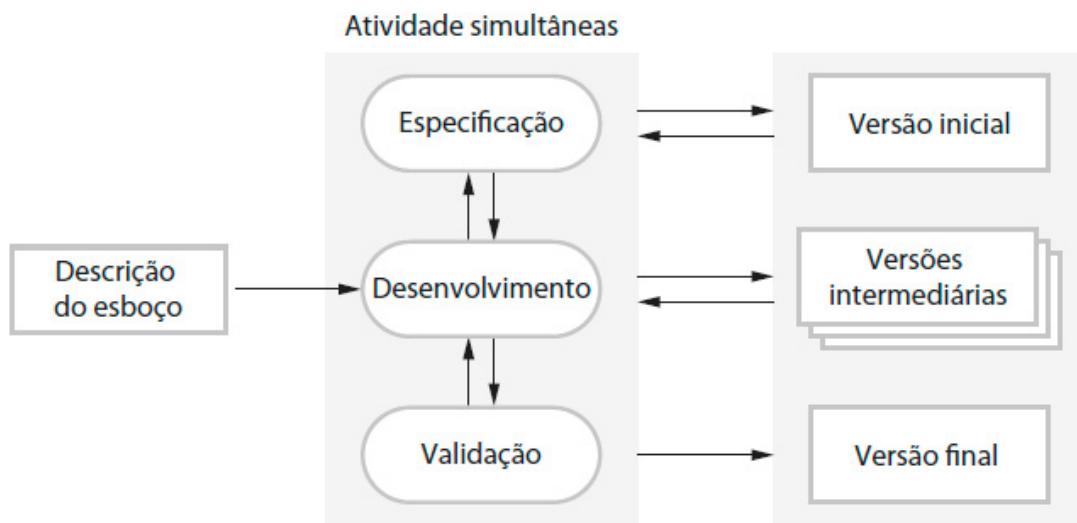


Figura. Desenvolvimento Incremental de acordo com Sommerville.

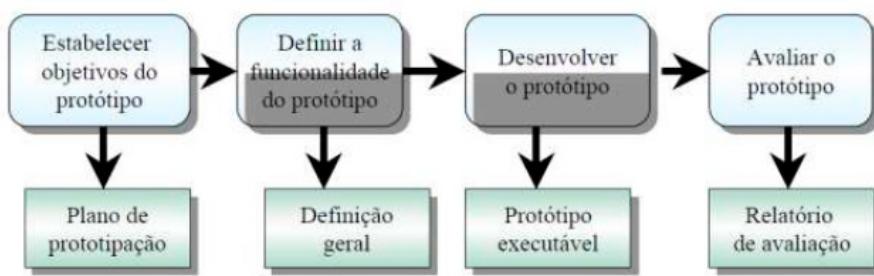
**Errado.**

### Prototipação (Chamada Também de Prototipagem)

O paradigma de **prototipagem** começa com a definição dos **requisitos**. Um projeto rápido é realizado e **concentra-se na representação daqueles aspectos que ficarão visíveis pelo cliente**. O protótipo é criado e avaliado e é ajustado para satisfazer as necessidades do cliente.

Idealmente, o protótipo serve como um mecanismo para identificação dos requisitos do software. A **prototipagem pode ser problemática**, pois o cliente vê o que parece ser uma versão executável do software, ignorando que o protótipo apenas consegue funcionar precariamente.

- Processo de desenvolvimento de protótipo:



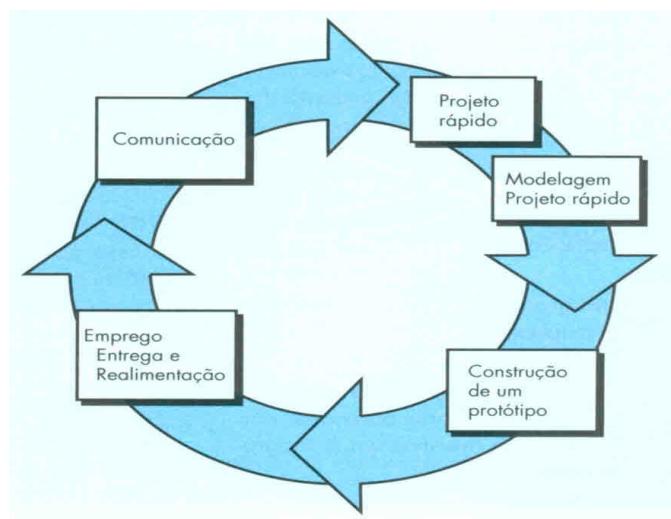
Nesse contexto:

- o cliente não possui uma visão clara de todos os requisitos da aplicação;
- o cliente quer avaliar a viabilidade de desenvolvimento da aplicação;
- o cliente alocará um usuário-chave no projeto, em tempo integral, a fim de que este possa participar ativamente de todas as fases do projeto;
- o cliente gostaria de ter uma versão preliminar do sistema, com base em uma versão inicial dos requisitos, ainda que isto demande um investimento inicial.

Em algumas situações, o cliente consegue definir apenas um conjunto de objetivos gerais do software, não identificando claramente seus requisitos. Em uma situação dessas, a **Prototipação (Prototipagem)** pode ser empregada em conjunto com outros modelos para auxiliar no entendimento do sistema.

Os objetivos do software são estabelecidos na comunicação com o cliente. A partir daí, um **protótipo descartável** é elaborado com o intuito de facilitar a compreensão do sistema por parte dos usuários.

Apesar da prototipagem poder ser aplicada como um modelo, em geral ela é mais utilizada como uma técnica para entendimento do sistema. A próxima figura apresenta o esquema deste modelo.



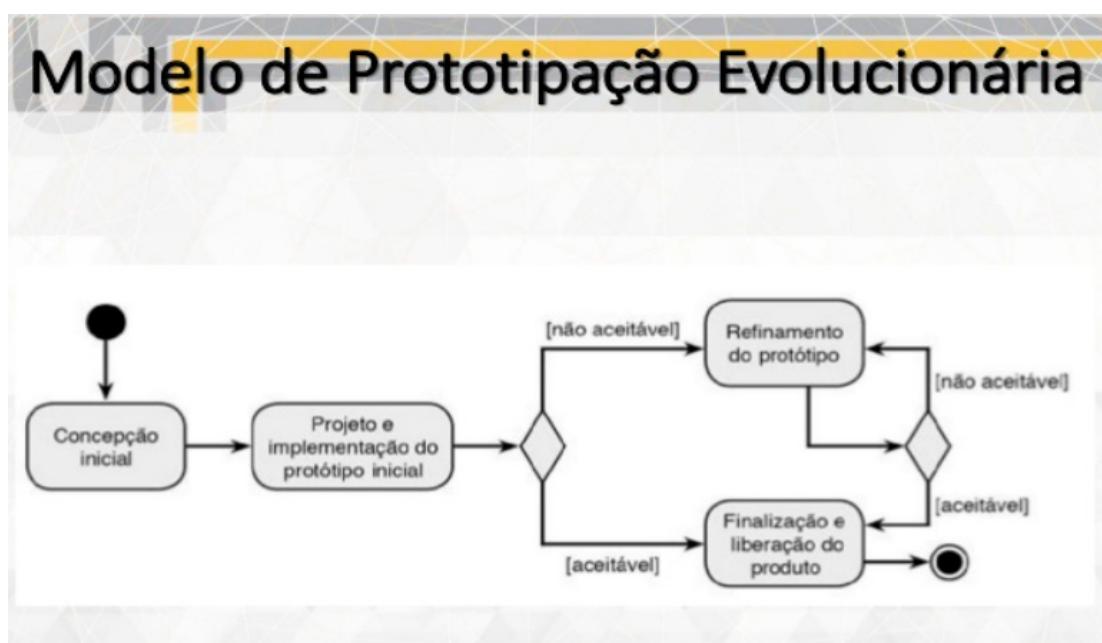
**Figura. O Paradigma da Prototipação. Fonte: Pressman (2011)**

- **Vantagens** da prototipagem:
  - Maior participação e comprometimento dos clientes e usuários; e
  - os resultados são apresentados mais rapidamente.
- Críticas:
  - **Forte dependência das linguagens e ambientes utilizados**, bem como da experiência da equipe;
  - **o cliente tende a considerar o protótipo como versão final**, podendo comprometer a qualidade do projeto; e
  - **o desenvolvedor tende a fazer concessões na implementação**, a fim de colocar um protótipo em funcionamento rapidamente. Estas concessões podem se tornar parte integrante do sistema.

**Veja a seguir os tipos de prototipação no processo de software:**

- -**Prototipação Evolucionária**

Uma abordagem para o desenvolvimento do sistema em que um protótipo inicial é produzido e refinado através de vários estágios até atingir o sistema final.



Fonte: <https://pt.slideshare.net/elainececiliagatto/modelos-de-processo-de-software-parte-3>

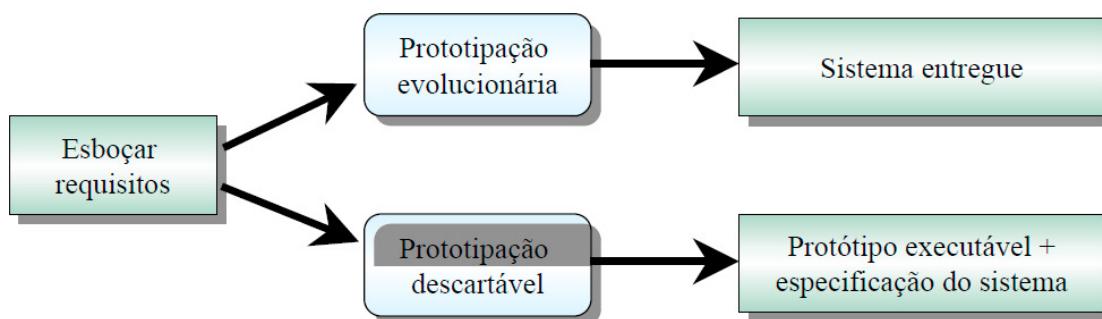
Na **Prototipação Evolucionária**, o protótipo evolui até se transformar no próprio produto entregue ao cliente, enquanto na **Prototipação Descartável** o protótipo é utilizado para esclarecer requisitos e avaliar riscos, sendo descartado ao final do processo.

O objetivo da prototipação evolucionária é fornecer aos usuários finais um sistema funcionando. O desenvolvimento começa com aqueles requisitos que são melhores compreendidos.

- **Prototipação Descartável:**

Um protótipo o qual é usualmente uma implementação prática do sistema é produzido para ajudar a levantar os problemas com os requisitos e depois descartado. O sistema é então desenvolvido usando algum outro processo de desenvolvimento.

O objetivo da prototipação descartável é validar ou derivar os requisitos do sistema. O processo de prototipação começa com aqueles requisitos que não são bem compreendidos.



**Obs.:** Pontos-chave:

Um protótipo de sistema pode ser usado para dar aos usuários finais uma impressão concreta das capacidades desse sistema.

A prototipação está se tornando cada vez mais comum para o desenvolvimento de sistema onde o desenvolvimento rápido é essencial.

Protótipos descartáveis são usados para a compreensão dos requisitos do sistema.

Na prototipação evolucionária, o sistema é desenvolvido pela evolução de uma versão inicial em uma versão final do sistema.

O desenvolvimento rápido é importante na prototipação de sistemas. Isso pode levar à exclusão de algumas funcionalidades do sistema ou na diminuição dos requisitos não funcionais.

Entre as técnicas de prototipação estão o uso de linguagens de nível muito elevado, a programação de bando de dados e a construção de protótipos a partir de componentes reutilizáveis.

A prototipação é essencial para o desenvolvimento de interfaces com o usuário, as quais são difíceis de serem especificadas usando um modelo estático. Os usuários deveriam estar envolvidos na avaliação e na evolução do protótipo.

## DIRETO DO CONCURSO

**003.** (CESPE/TCE-RN/2009) A prototipação, uma abordagem para desenvolvimento de software na qual se cria um modelo do software que será implementado, é composta de quatro etapas: planejamento, análise de risco, engenharia e avaliação do cliente.



O modelo de prototipação pode contemplar as etapas listadas a seguir:

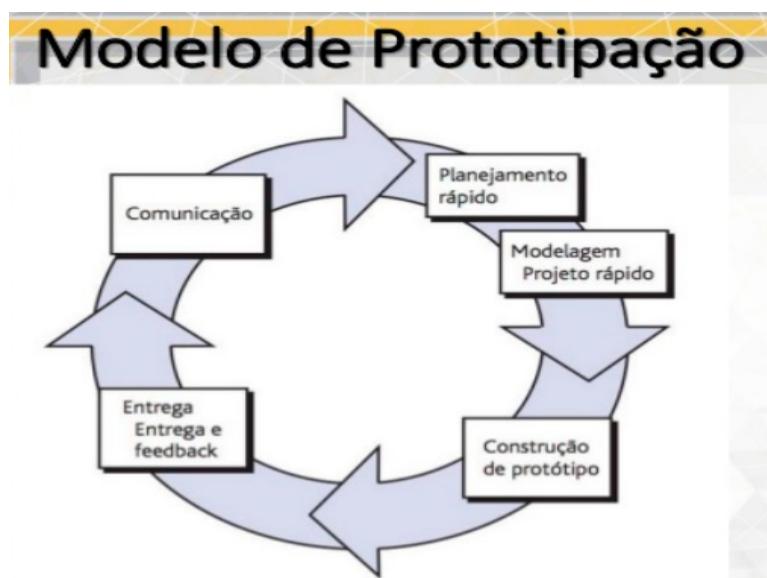


Figura. Esquema do Modelo de Prototipação.

Fonte: <https://pt.slideshare.net/elainececiliagatto/modelos-de-processo-de-software-parte-3>

<b>Comunicação:</b>	<b>Planejamento Rápido:</b>	<b>Modelagem Projeto Rápido:</b>
<ul style="list-style-type: none"> <li>• Reunião com os envolvidos</li> <li>• Definição dos objetivos gerais do software</li> <li>• Identificação dos requisitos</li> <li>• Esquematização das áreas que precisam de definição mais ampla</li> </ul>	<ul style="list-style-type: none"> <li>• Após a etapa de comunicação, uma iteração de prototipação é planejada rapidamente e ocorre a modelagem na forma de um projeto rápido</li> </ul>	<ul style="list-style-type: none"> <li>• Concentra-se em uma representação dos aspectos do software que serão visíveis para os usuários</li> </ul>

<b>Construção de Protótipo:</b>	<b>Entrega e Feedback</b>
<ul style="list-style-type: none"> <li>• Construção do protótipo propriamente dito</li> </ul>	<ul style="list-style-type: none"> <li>• O protótipo é entregue para todos os envolvidos e avaliados por eles</li> <li>• O feedback é usado para refinar ainda mais os requisitos</li> </ul>

Na **Prototipação Evolucionária**, o protótipo evolui até se transformar no próprio produto entregue ao cliente, enquanto na **Prototipação Descartável** o protótipo é utilizado para esclarecer requisitos e avaliar riscos, sendo descartado ao final do processo.

**Errado.**

**004. (FGV/2014/CM-RECIFE/ANALISTA LEGISLATIVO – ANALISTA DE SISTEMAS)**

Protótipos auxiliam na elicitação e na validação dos requisitos de sistemas computacionais.

Duas das técnicas muito utilizadas durante uma prototipação são:

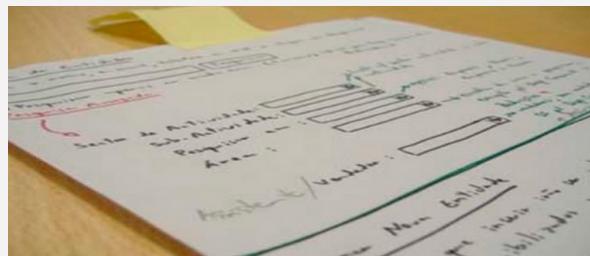
- a) prototipação em papel e etnografia;
- b) etnografia e wireframes;
- c) prototipação em papel e wireframes;
- d) mockups e pesquisa de mercado;
- e) entrevistas e pesquisa de mercado.



Técnicas de prototipação:

Utiliza meios físicos como papel, cartolina e papelão para a criação de objetos que irão demonstrar como será o objeto final desejado.

### Prototipação em papel



É o documento que apresenta a estrutura e o conteúdo da interface, indicando o peso e a relevância de cada elemento do layout e sua relação com os demais elementos formadores do todo.

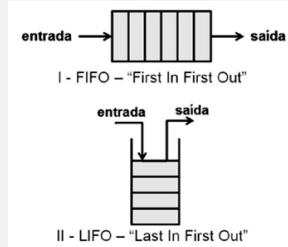
Este wireframe não especifica design gráfico.  
Sua função é apresentar os elementos que vão compor a página.



## Wireframe

É uma peça do tamanho real ou exagerado do produto. Apresenta o fluxo de uma interação com a interface, e, além disto, possui elementos como botões e menus entre outros objetos possibilitando ao usuário interagir com a interface.

## Mock-ups



A técnica de **estudos etnográficos** (ou **etnografia**) é proveniente das disciplinas de Antropologia, que consiste no estudo de um objeto por vivência direta da realidade onde este se insere. O objetivo deste estudo é identificar o modo como realmente as pessoas executam as suas funções que, muitas vezes, difere da forma como as definições dos processos sugerem que elas devem fazer; como ocorre a cooperação e conhecimento das atividades entre as pessoas. Dentre as assertivas, considerar **prototipação em papel e wireframes**.

Veja mais: <http://pt.slideshare.net/leopp/prototipao-de-software>

**Letra c.**

## Modelo Espiral

No modelo Espiral, assume-se que o processo de desenvolvimento ocorre em **ciclos**, cada um contendo fases de avaliação e planejamento em que a opção de abordagem para a próxima fase (ou ciclo) é determinada.

Nesse modelo acrescenta-se a **Análise dos Riscos** ao ciclo de vida para auxiliar as decisões a respeito da próxima iteração. A figura seguinte apresenta o esquema desse modelo.

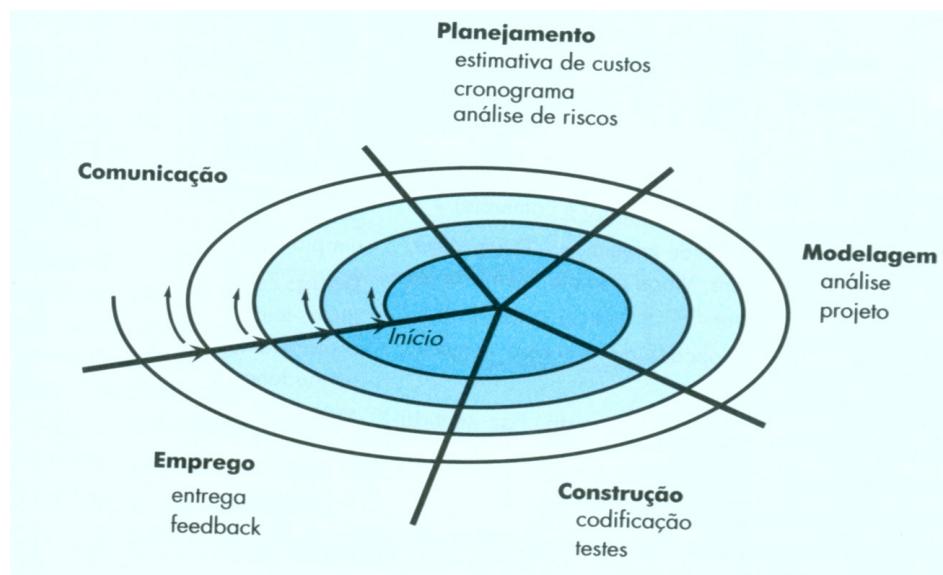


Figura. O Modelo Espiral Típico. Fonte: Pressman (2011)

**Obs.:** No **modelo de processo de desenvolvimento em espiral**, cada loop na espiral representa uma fase do processo de software. Esse modelo exige a consideração direta dos **riscos técnicos** em todos os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles se tornem problemáticos.

Apesar desse modelo reunir as melhores características dos outros, algumas **críticas** podem ser feitas:

- exige gerentes e técnicos experientes;
- uma vez que o modelo em espiral pode levar ao desenvolvimento em paralelo de múltiplas partes do projeto, as tarefas gerenciais para acompanhamento e controle do projeto são mais complexas;
- é necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

## DIRETO DO CONCURSO

**005. (CESPE/TRT – 10ª REGIÃO (DF E TO)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO/2013)** Tendo em vista que o desenvolvimento de um **software** compreende

várias fases, que vão desde a definição básica até o uso do *software*, e que, nesse processo, diversos modelos, métodos e procedimentos de construção podem ser utilizados, julgue os itens subsecutivos.

No ciclo de vida da primeira versão do modelo em espiral, a etapa de análise de riscos é realizada dentro da fase de desenvolvimento.



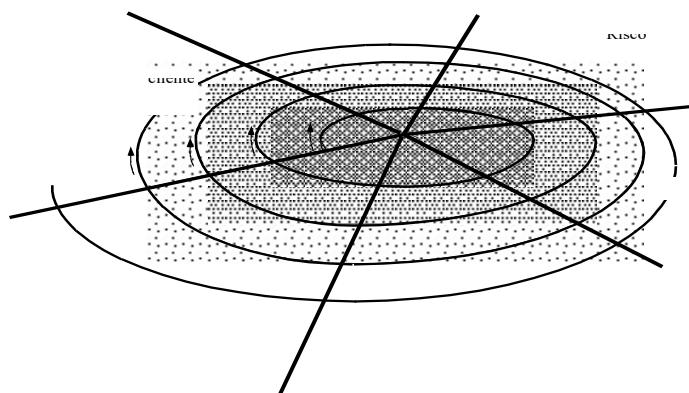
Modelo em Espiral exige a consideração direta dos riscos técnicos em TODOS os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles se tornem problemáticos.

**Certo.**

**006.** (CESPE/MPE-RN/GESTÃO E ANÁLISE/2010) O modelo em espiral difere principalmente dos outros modelos de processo de software por

- a) não contemplar o protótipo.
- b) reconhecer explicitamente o risco.
- c) não ter fases.
- d) possuir uma fase evolucionária.
- e) não contemplar o projeto do produto.

No modelo em espiral acrescenta-se a Análise dos Riscos ao ciclo de vida para auxiliar as decisões a respeito da próxima iteração. A figura seguinte apresenta o esquema deste modelo.



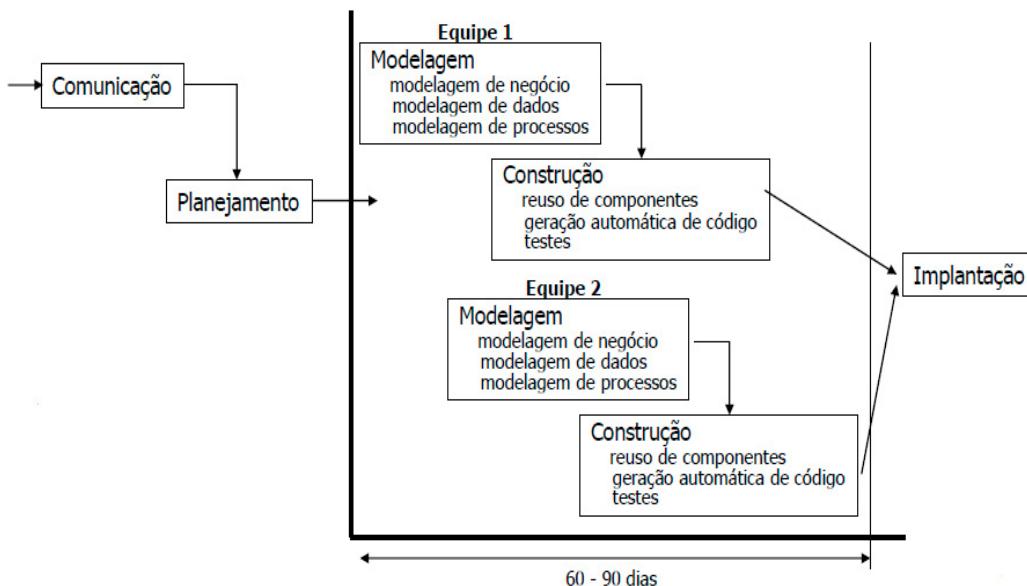
**Obs.:** Observe que cada *loop* na espiral representa uma fase do processo de *software*. Esse modelo exige a consideração direta dos riscos técnicos em todos os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles se tornem problemáticos.

**Letra b.**

## RAD (Rapid Application Development – Desenvolvimento Rápido de Aplicação)

Trata-se de um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto. É uma adaptação “de alta velocidade” do modelo em cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes.

- **Desvantagens:** projetos grandes precisam de muitas equipes; exige comprometimento dos clientes e desenvolvedores; sistemas não modularizados são problemáticos; não é adequado para projetos com grandes riscos técnicos.



## RUP (Rational Unified Process)

O **Rational Unified Process (RUP)** é um exemplo de modelo de processo de desenvolvimento baseado no *Unified Process* (Processo Unificado) desenvolvido pela Rational.

O RUP oferece uma abordagem baseada em **disciplinas** para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir a produção de software de alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis.

O **Rational Unified Process** é um processo de desenvolvimento **iterativo** e **incremental**, no qual o software não é implementado em um instante no fim do projeto, mas é, ao contrário, desenvolvido e implementado em partes. A cada iteração deste processo utiliza-se **quatro fases**, a saber: **Concepção, Elaboração, Construção e Transição**.

- Durante a **Concepção ou Iniciação (Inception)**, estabelece-se a lógica do domínio da aplicação para o projeto e decide o escopo do projeto. É aqui que se obtém

o comprometimento do patrocinador do projeto para seguir adiante. Nesta fase compreende-se o problema da tecnologia empregada por meio da definição dos *use cases* mais críticos. Define-se aqui o caso de negócio e escopo do projeto.

- Na **Elaboração (Elaboration)** coleta-se requisitos mais detalhados, faz uma análise e um projeto de alto nível para estabelecer uma arquitetura básica, e cria um plano para a construção do sistema. Deve-se analisar o domínio do problema, estabelecer a arquitetura, desenvolver o plano do projeto e eliminar elementos de alto risco.
- A fase de **Construção (Construction)** consiste de várias **iterações**, nas quais cada iteração constrói software de qualidade de produção, testado e integrado, que satisfaz um subconjunto de requisitos de projeto. Cada iteração contém todas as fases usuais do ciclo de vida da análise, do projeto, da implementação e do teste. Desenvolve-se o software e prepara-se o mesmo para a transição para os usuários. Além do código, também são produzidos os casos de teste e a documentação.
- Mesmo com este tipo de processo iterativo, algum trabalho tem que ser deixado para ser feito no fim, na fase de **Transição (Transition)**. Nesta fase são realizados os treinamentos dos usuários e a transição do produto para utilização. Este trabalho pode incluir também testes beta e ajuste de performance.

Os projetos variam de acordo com o volume de formalidade que eles têm. Projetos de muita formalidade têm muitos documentos formais a serem entregues, reuniões formais, encerramentos formais. Projetos de pouca formalidade podem ter uma fase de concepção que consiste de um bate-papo de uma hora com o patrocinador do projeto e um plano que cabe em uma folha de papel. Naturalmente quanto maior o projeto, mais formalidade precisa. O fundamental de cada fase ainda acontece, mas de formas bem diferentes.

Após a **fase de Transição** de uma iteração completa, o produto pode voltar a percorrer cada uma das fases para se produzir uma nova versão do produto.

**Obs.:** **Cada uma das quatro fases do RUP é dividida em iterações, em que cada uma delas é um ciclo completo de desenvolvimento resultando em uma versão de um produto executável que constitui um subconjunto do produto final. Cada iteração é organizada em fluxos de trabalho (workflows) de processo, com uma ênfase diferente em cada fase.**

Os fluxos de trabalho de processo do RUP são:

- Gerenciamento de Projeto (*Project Management*);
- Modelagem Comercial ou de Negócio (*Business Modeling*);
- Requisitos ou Exigências (*Requirements*);
- Análise e Projeto (*Analysys & Design*);

- Implementação (*Implementation*);
- Testes (*Test*);
- Distribuição ou Entrega (*Deployment*);
- Gerência de Configuração e Mudanças (*Configuration and Change Management*); e
- Ambiente (*Environment*).

A figura ilustrada a seguir apresenta o relacionamento entre as fases e o esforço de desenvolvimento em cada fluxo de trabalho de processo.

**Obs.:** O RUP é composto por **nove** disciplinas e **quatro** fases!

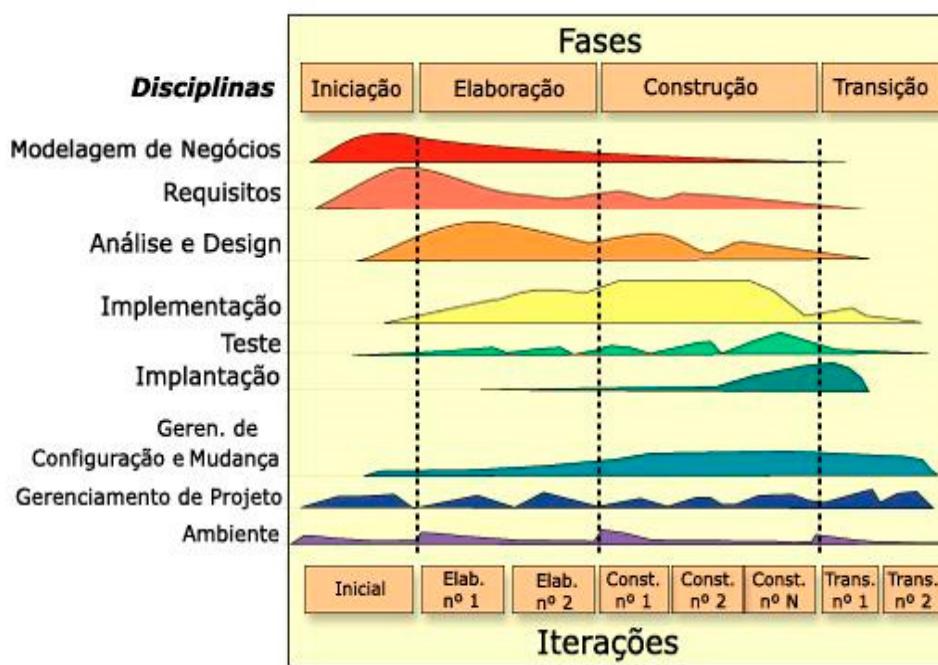


Figura. Fases e disciplinas do RUP. Fonte: (KRUCHTEN, 2003)

Na figura, o **eixo horizontal** representa o tempo e mostra os aspectos do ciclo de vida do processo como se desdobra. Já o **eixo vertical** representa os fluxos essenciais do processo, que agrupa logicamente as atividades por natureza.

## Desenvolvimento Baseado em Componentes

Desenvolve aplicações a partir de componentes de software pré-empacotados (faz **reuso** de componentes de softwares já existentes, para conseguir redução no tempo do ciclo de desenvolvimento, bem como redução no custo do projeto) (Pressman, 2011).

## Document-Driven Programming (DDP)

Uma prática de programação em que se procura ao máximo documentar todo o planejamento e descrições de funcionalidades para os blocos de código a serem construídos, e em torno dessa documentação o código vai evoluindo. Na utilização com XML, por exemplo, consiste em referenciar na documentação componentes de interface de usuário e componentes de lógica de negócios necessários para o funcionamento da aplicação.

## Test-Driven Programming (TDP)

O TDP (*Test Drive Programming*) nos leva às definições e práticas do próprio **TDD(Test-Driven Development, ou Desenvolvimento Guiado por Testes, ou simplesmente TDD)**, que consiste em uma técnica de desenvolvimento de software em que primeiro são criados os testes e somente **depois é escrito o código necessário para passar por eles**. Veja mais em: <http://www.devmedia.com.br/test-driven-development-tdd/10025#ixzz3S7zcqo9p>

### DIRETO DO CONCURSO

**007.** (CESPE/2013/MPOG/TECNOLOGIA DA INFORMAÇÃO) Ao realizar o TDD (test-driven development), o programador é conduzido a pensar em decisões de design antes de pensar em código de implementação, o que cria um maior acoplamento, uma vez que seu objetivo é pensar na lógica e nas responsabilidades de cada classe.



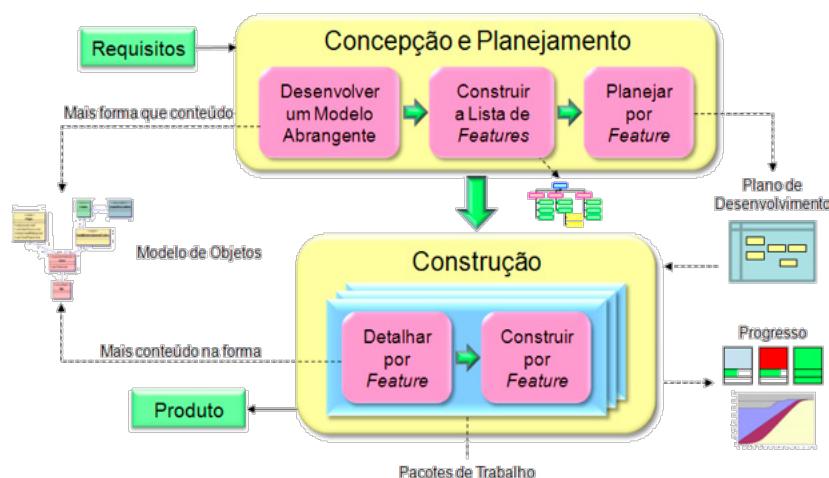
**TDD (Test-Driven Development)** procura entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. Cria-se com o TDD o mínimo de acoplamento. Segundo destaca <http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/>, com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos. **Errado.**

## Desenvolvimento Guiado por Funcionalidades (FDD – Feature Driven Development)

Esse modelo enfatiza o uso de orientação a objetos e possui apenas duas grandes fases:

### 1 – Concepção e Planejamento e 2 – Construção.

A estrutura básica do FDD é apresentada na figura seguinte, obtida em [http://pt.wikipedia.org/wiki/Feature\\_Driven\\_Development](http://pt.wikipedia.org/wiki/Feature_Driven_Development)



Conforme a figura acima, a fase de Concepção e Planejamento divide-se em 3 processos:

- **Desenvolver um Modelo Abrangente:** visa o entendimento das regras de negócio do projeto para gerar um modelo de objetos;
- **Construir uma Lista de Funcionalidades (Features):** divisão do modelo desenvolvido em três camadas: áreas de negócio, atividades de negócio e passos automatizados da atividade (funcionalidades). O resultado é uma hierarquia de funcionalidades do produto (product backlog); e
- **Planejar por Funcionalidade:** a estimativa da complexidade e dependência das funcionalidades, gerando um plano de desenvolvimento.

Já a Construção, divide-se em duas disciplinas, a saber: Detalhar por Funcionalidade: detalhar os requisitos e outros artefatos para a codificação; e Construir por Funcionalidade: o código e desenvolvido, incrementando-se o projeto.

## Modelo de Métodos Formais

Conforme destaca Pressman (2011, p. 69), esse modelo **engloba um conjunto de atividades que conduzem à especificação matemática formal do software**. Os métodos formais possibilitam especificar, desenvolver e verificar um sistema baseado em computador por meio da aplicação de uma notação matemática rigorosa.

Quando são utilizados os métodos formais durante o desenvolvimento, estes oferecem um mecanismo que elimina muitos dos problemas que são difíceis de ser superados com o uso de

outros paradigmas de engenharia de software. Ambiguidade, incompletude e inconsistência podem ser descobertas e corrigidas mais facilmente devido à aplicação de análise matemática (Pressman, 2011).

Mais características (Pressman, 2011):

- Atualmente, o desenvolvimento de métodos formais consome muito tempo e dinheiro;
- Número limitado de desenvolvedores com formação para aplicação de métodos formais, sendo necessário treinamento extensivo;
- É difícil usar os modelos como um meio de comunicação com clientes tecnicamente despreparados.

Apesar disso, tem conquistado adeptos entre os desenvolvedores de software, que necessitam desenvolver softwares com fator crítico de segurança (como softwares para aeronaves e equipamentos médicos), e entre os desenvolvedores que sofreriam pesadas sanções econômicas se ocorressem erros no software (Pressman, 2011).

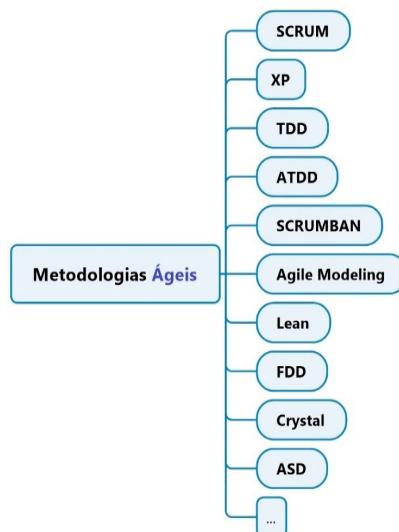
## Processo Ágil

A Engenharia de Software vem há anos criando técnicas de modelagem, projeto e desenvolvimento de sistemas. Dentre os desafios dos pesquisadores da área, pode-se citar a preocupação em desenvolver softwares com qualidade garantida, no prazo estabelecido e sem alocar recursos além do previsto. No entanto, muitas das metodologias criadas são consumidoras de muitos recursos, aplicando-se principalmente a grandes sistemas.

Como a Engenharia de Software ainda está evoluindo, novos modelos estão sendo apresentados, como: **desenvolvimento ágil**; dentre outros.

Os modelos ágeis de desenvolvimento de software têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos do desenvolvimento.

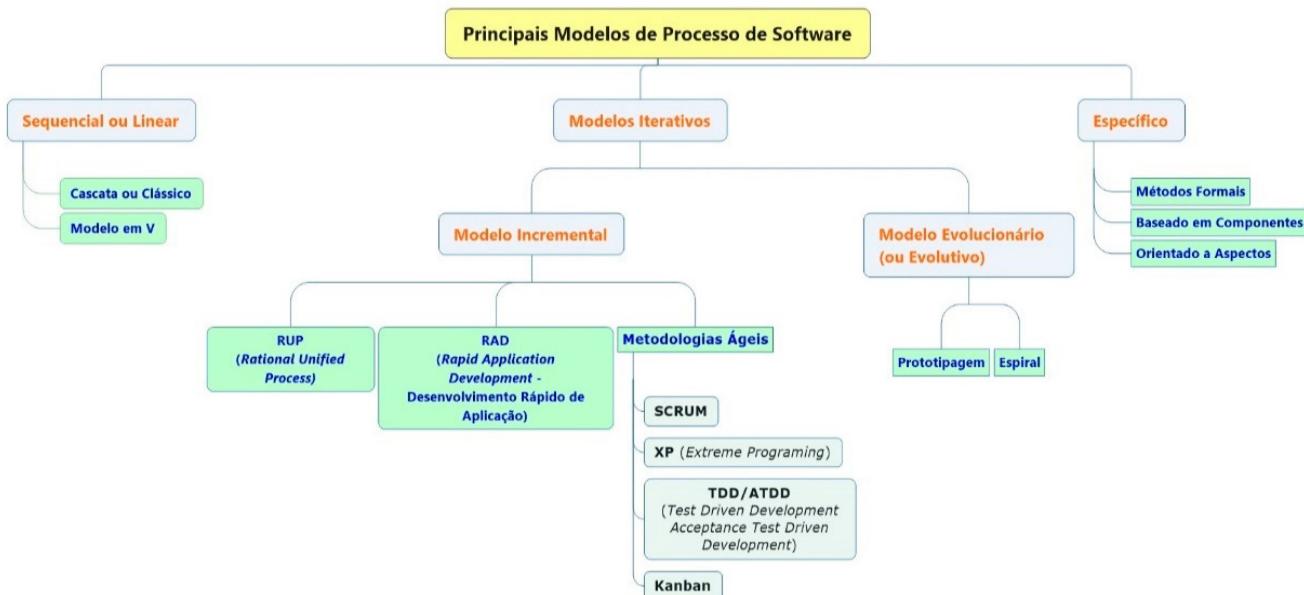
São **exemplos** de metodologias ágeis:



**Figura. Metodologias Ágeis. Fonte: Quintão (2021)**

Aqui merecem destaque as metodologias ágeis de desenvolvimento, como SCRUM, eXtreme Programming (XP), TDD, BDD, DDD, Kanban e Scrumban.

Veja a seguir um esquema destacando as Metodologias Ágeis no âmbito dos principais modelos de processo de software.



**Figura. Principais Modelos de Processo de Software. Fonte: Quintão (2021)**

Veja a seguir algumas características fundamentais entre as metodologias ágeis, segundo Sommerville:

- **não** há especificação detalhada do sistema, e a documentação do projeto é minimizada ou gerada automaticamente;
- o sistema é desenvolvido em uma série de versões;
- interfaces de usuário do sistema são geralmente desenvolvidas com um sistema **interativo** de desenvolvimento que permite a criação rápida do projeto.

## 4.2. VISÃO GERAL SOBRE O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Há vários processos de desenvolvimento propostos.

Bezerra (2007) destaca que é um consenso na comunidade de desenvolvimento de software o fato de que **não existe o melhor processo de desenvolvimento**, aquele que melhor se aplica a todas as situações de desenvolvimento.

Cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento. Entretanto, podem-se distinguir atividades que, com uma ou outra modificação, são **comuns à maioria dos processos existentes**. São elas:

## Modelagem de Negócio (Modelagem de Processos de Negócio, Análise do Domínio ou Análise do Negócio)

Bezerra (2007, p. 28) destaca que na análise do domínio **um primeiro objetivo é identificar e modelar os objetos do mundo real** que, de alguma forma, serão processados pela aplicação em desenvolvimento. Como exemplo Bezerra cita que um aluno é um objeto do mundo real que um sistema de controle acadêmico deve processar. Dessa forma, aluno é um objeto do domínio.

Uma característica da análise do domínio é que **os objetos identificados fazem sentido para os especialistas do domínio**, por conta de corresponderem a conceitos com os quais esses profissionais lidam diariamente em seu trabalho. Por isso, segundo Bezerra (2007), na análise de domínio, os analistas devem interagir com os especialistas do domínio. Outros exemplos de objetos do domínio Instituição de Ensino são: professor, disciplina, turma, sala de aula, etc.

Outro objetivo da análise de domínio, mencionado por Bezerra (2007) é identificar as regras do negócio e os processos do negócio realizados pela organização.

## Engenharia de Requisitos

É o uso sistemático de **princípios, técnicas, linguagens e ferramentas** comprovadas para análise, documentação, evolução continuada das necessidades dos usuários e especificação do comportamento externo de um sistema para satisfazer as necessidades do usuário, que sejam efetivas em termos de custos. Visa, principalmente, **o entendimento escrito do problema**.

Algumas considerações importantes:

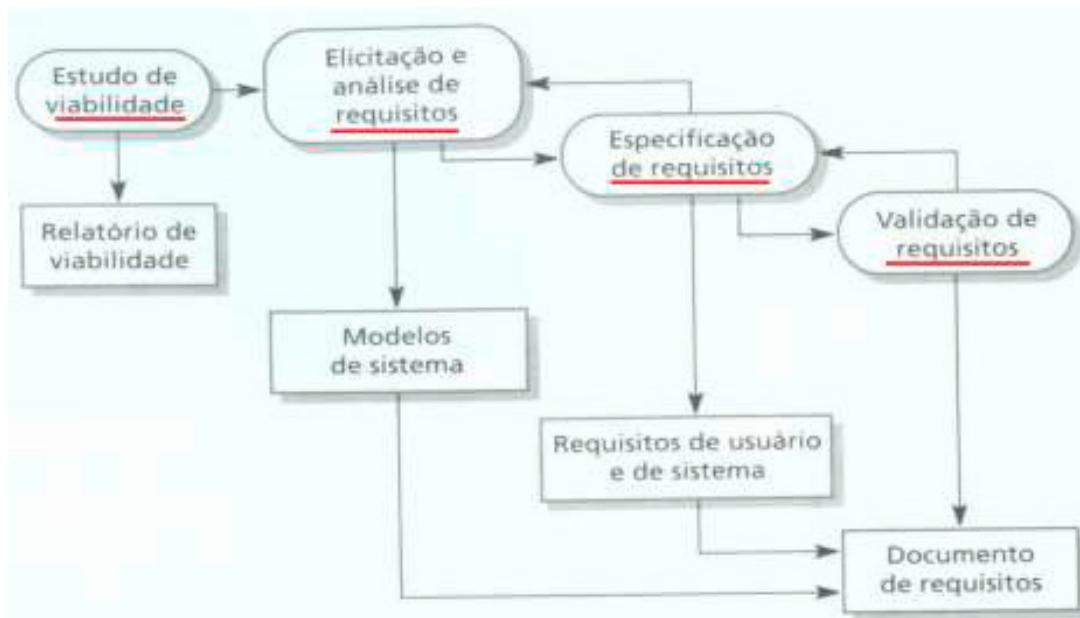
- é uma **abordagem sistemática**, ou seja, constituída por um conjunto de processos estruturados para extrair, validar e manter os requisitos de um sistema;
- constitui a ponte entre a comunicação com o cliente, a documentação gerada, o projeto e o desenvolvimento.

**Obs.:** | O objetivo do processo de engenharia de requisitos é criar e manter um documento de requisitos de sistema.

O processo de **Engenharia de Requisitos**, seguindo a abordagem do Sommerville (2007), inclui:

- **estudo de viabilidade;**
- **elicitação e análise de requisitos;**
- **especificação de requisitos;**
- **validação de requisitos;**
- **gerenciamento de requisitos.**

Sommerville (2007) destaca que esses subprocessos estão relacionados à avaliação de se o sistema é útil para a empresa (**estudo de viabilidade**), obtenção de requisitos (**elicitação e análise**), conversão desses requisitos em alguma forma-padrão (**especificação**) e verificação de se os requisitos realmente definem o sistema que o cliente deseja (**validação**).



**Figura. Processo de Engenharia de Requisitos – Fonte: Sommerville (2007)**

A Engenharia de Requisitos, segundo Pressman (2011), é composta por 7 tarefas distintas, que são:

- Concepção;
- Levantamento;
- Elaboração;
- Negociação;
- Especificação;
- Validação;
- Gestão de Requisitos.

## Análise

Pode-se definir “análise” como o processo de decompor o todo em suas partes componentes, examinando estas partes, conhecendo sua natureza, funções e relações.

A tarefa de construir **Sistemas de Informação** é bastante complexa, configurandose, na realidade em um processo de solução de problemas. Neste sentido, dizemos que a análise de sistemas consiste nos métodos e técnicas de avaliação e especificação da solução de problemas, para implementação em algum meio que a suporte, utilizando mecanismos apropriados. É uma etapa no processo de desenvolvimento de software.

Segundo Pressman todos os métodos de análise devem ser capazes de suportar 5 atividades:

- representar e entender o domínio da informação;
- definir as funções que o software deve executar;
- representar o comportamento do software em função dos eventos externos;
- particionar os modelos de informação, função e comportamento de maneira a apresentar os detalhes de forma hierárquica;
- prover a informação essencial em direção à determinação dos detalhes de implementação.

Para que a análise seja bem feita devemos não só entender o que vamos fazer, mas também desenvolver uma representação que permita que outros entendam o que é necessário para que o sistema que está sendo desenvolvido atinja sua finalidade.

A grande maioria dos autores advoga que não devemos levar em conta a tecnologia que empregaremos durante a análise de sistemas.

**Obs.:** | A análise deve se preocupar com “o que fazer” e nunca com o “como fazer”. Para isso, fazemos a modelagem dos sistemas, utilizando abstrações.

## Projeto (Desenho)

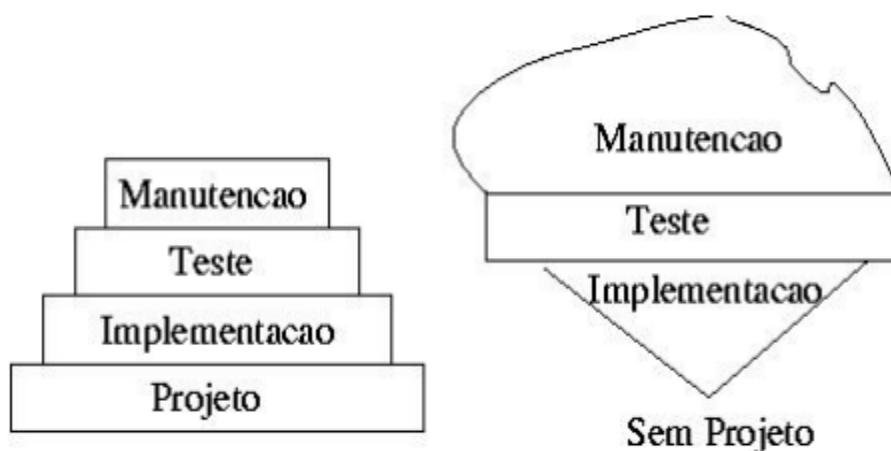
Bezerra (2007) destaca que o foco principal da ANÁLISE são os aspectos lógicos e independentes de implementação de um sistema (os requisitos).

Já **NA FASE DE PROJETO**, determina-se “**COMO**” o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes (a fase de projeto considera os aspectos físicos e dependentes de implementação).

Aos modelos construídos na fase de análise são adicionadas as restrições de tecnologia (Bezerra, 2007). Como exemplos de aspectos a serem considerados na fase de projeto temos: arquitetura do sistema, padrão de interface gráfica, a linguagem de programação, o gerenciador de banco de dados, etc.

Essa fase produz uma descrição computacional do que o software deve fazer e deve ser coerente com a descrição feita na análise. Em alguns casos, algumas restrições da tecnologia a ser utilizada já foram amarradas no levantamento dos requisitos. Em outros casos, essas restrições devem ser especificadas. Mas, em todos os casos, a fase de projeto do sistema é direcionada pelos modelos construídos na fase de análise e pelo planejamento do sistema (Bezerra, 2007).

O projeto servirá como fundamento para as fases de codificação, teste e manutenção.



## Implementação

Nessa fase o sistema é codificado, ou seja, ocorre a tradução da descrição computacional obtida na fase de projeto em código executável mediante o uso de uma ou mais linguagens de programação (Bezerra, 2007).

Em um processo de desenvolvimento orientado a objetos, a implementação envolve a criação do código-fonte correspondente às **classes** de objetos do sistema utilizando linguagens de programação como C++, Java etc. Além da codificação desde o início, a implementação pode também reutilizar componentes de software, dentre outros (Bezerra, 2007).

<b>Desenvolvimento Estruturado:</b>	<b>Desenvolvimento Orientado a Objetos:</b>
<ul style="list-style-type: none"> <li>• Fluxo de dados</li> <li>• Transformações</li> <li>• Repositórios de dados</li> <li>• Entidades</li> <li>• Especificação de procedimentos</li> <li>• Dicionário de dados</li> </ul> <p><b>Foco nos processamentos do sistema.</b></p>	<ul style="list-style-type: none"> <li>▪ Objetos (Classes)                             <ul style="list-style-type: none"> <li>▪ Atributos</li> <li>▪ Operações</li> </ul> </li> <li>▪ Associações e Multiplicidade</li> <li>▪ Herança</li> <li>▪ Outros componentes especializados</li> </ul> <p><b>Foco nos componentes do sistema.</b></p>

É importante observar o foco de cada desenvolvimento. No Desenvolvimento Estruturado, o foco é o processamento dos sistemas, já no Desenvolvimento Orientado a Objetos os componentes do sistema são priorizados. Isto é diferente da programação convencional, em que a estrutura e o comportamento dos dados têm poucos vínculos entre si.

## Testes

Diversas atividades de teste são realizadas para verificação do sistema construído, levando-se em conta a especificação feita na fase de projeto. O principal produto dessa fase é o **relatório de testes**, que contém informações sobre erros detectados no software. Após a atividade de testes, os diversos módulos do sistema são integrados, resultando finalmente no produto de software (Bezerra, 2007).

Antes de continuar, gostaria de reforçar o entendimento de que **o teste NÃO é uma fase que deve ser executada ao final do projeto, mas uma atividade que deve ser exercida ao longo do processo de desenvolvimento do software.**

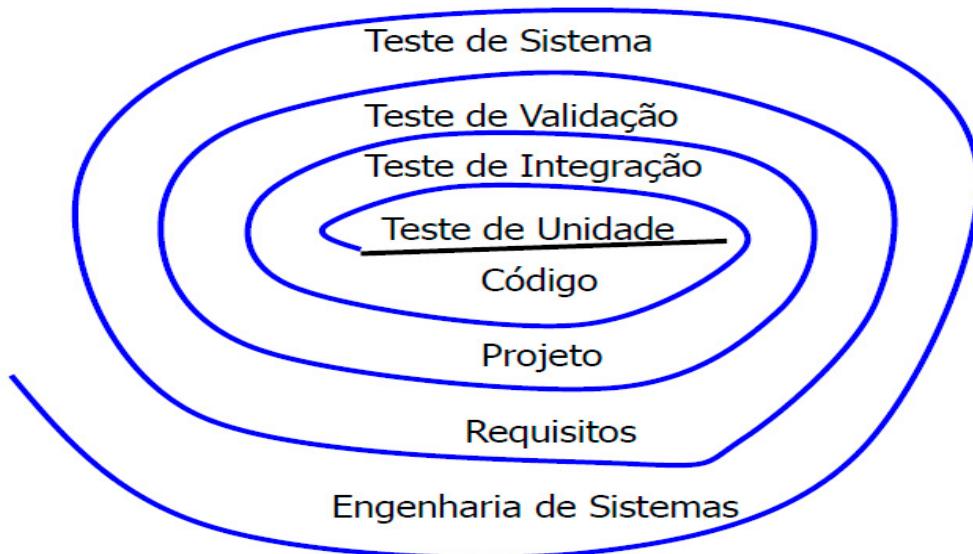
**Teste de software** é uma atividade cara que exige tempo, planejamento, conhecimento técnico, infraestrutura e comprometimento.

Por maior e mais organizado que seja o esforço para a realização das atividades de testes, é impossível garantir 100% de cobertura de todos os requisitos ou linhas de código existentes no sistema.

Edsger Dijkstra sabiamente resumiu este fato com a seguinte frase: “**Testes podem mostrar a presença de erros, mas não a sua ausência**”. Dessa forma, está em suas mãos a responsabilidade de conhecer e aplicar as melhores práticas na gestão de testes para garantir a maior cobertura com o mínimo de esforço.

Algumas observações:

- **Teste de Integração:** as equipes de testes têm acesso ao código-fonte do sistema. Quando um problema é descoberto, a equipe de integração tenta encontrar a origem do problema e identificar os componentes que devem ser depurados. Os testes de integração geralmente estão relacionados à descoberta de defeitos no sistema (SOMMERVILLE, 2007).
- **Teste de Unidade:** os pequenos elementos passíveis de teste são testados individualmente, geralmente ao mesmo tempo em que são implementados (KRUCHTEN, 2003).
- **Testes “Caixa-Preta” (Black Box):** analisam a funcionalidade e a aderência aos requisitos, em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado;
- **Teste de Caixa Branca:** é uma abordagem para projetar casos de teste na qual os testes são derivados do conhecimento da estrutura e da implementação do software. O entendimento do algoritmo usado em um componente pode ajudar a identificar partições e casos de teste adicionais (SOMMERVILLE, 2007). **O teste “caixa-branca” avalia a lógica interna do sistema!**



**Figura. Estratégia de Teste (Pressman, 2011)**

## Implantação

Nessa fase **o sistema é empacotado, distribuído e instalado no ambiente do usuário.**

Os manuais do sistema são escritos, os arquivos são carregados, os dados são importados para o sistema, e os usuários treinados para utilizar o sistema corretamente.

Em alguns casos, aqui também ocorre a migração de sistemas de software e de dados preexistentes (Bezerra, 2007).

## Homologação

É a etapa em que o cliente deverá testar as funcionalidades do sistema que serão posteriormente colocadas em produção ou refeitas.

## Manutenção

**Processo de melhoria e otimização de um software já desenvolvido** (versão de produção), como também reparo de defeitos.

### Tipos de manutenção de software:

- **Corretiva:** faz correção de erros encontrados na verificação ou na validação;
- **Adaptativa:** adaptação a mudanças externas;
- **Melhoria (perfectiva):** melhorias requeridas pelos usuários;
- **Preventiva** ou de reengenharia: Abordagem proativa com foco na melhoria da manutibilidade.

## 5. MODELOS DE GESTÃO: CATEDRAL E BAZAR

### 5.1. CATEDRAL (TRADICIONAL)

- **Metodologia de desenvolvimento típica de software proprietário**, baseado na concepção tradicional de desenvolvimento de software.
- O código fonte é avaliado em cada atualização do software, mas **o código desenvolvido entre as duas atualizações é restrito ao grupo de desenvolvedores**.
- **Desenvolvimento** de software de forma **controlada e centralizada**;
- **Necessita de um arquiteto central**, que leva toda a responsabilidade do processo de desenvolvimento.

### 5.2. BAZAR (COLABORATIVO)

- **Metodologia caracterizada pelo desenvolvimento do software de maneira totalmente aberta e disseminada**.
- Parte do princípio de que qualquer pessoa pode contribuir para o desenvolvimento de um programa e as melhores contribuições serão mantidas no código. Uma de suas principais características é que problemas são facilmente encontrados, pois como citado por Eric Raymond, “com muitos olhos, todos os ‘bugs’ (erros) serão visíveis”.
- Tem um objetivo claro e de fácil motivação;
- Tem bom líder com missão de manter a comunidade motivada, não deixando que o objetivo se perca e busca tomar as medidas necessárias para que tudo funcione;
- Tem uma comunidade de participantes, trabalhando de forma totalmente descentralizada;
- Tem um ambiente de trabalho dinâmico, que fornece meios de comunicação e disseminação do código (como a Internet e ferramentas como sites e listas de discussões).

A tabela seguinte destaca as principais diferenças entre o **Modelo Catedral** e o **Modelo Bazar**.

Modelo Catedral	Modelo Bazar
Trabalho controlado por um grupo de projetistas	Não há hierarquia entre os participantes, apenas um líder
Estabelecimento de metodologias, tarefas e prazos	Projetos informalmente organizados ao redor de uma proposta inicial
Demora para lançamento de atualizações	Atualizações constantes
Dificuldade para atingir <i>massa crítica</i>	Participação voluntária e consequentemente <i>massa crítica</i> atingida rapidamente
Dificuldade para atingir qualidade esperada	Qualidade atingida rapidamente

**Figura. Principais diferenças entre o Modelo Catedral e o Modelo Bazar**

O conteúdo deste livro eletrônico é licenciado para 61984693488 Martins Rodrigues - 00193743132, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

O livro “The Cathedral and the Bazaar”, de Eric Raymond, mostrou que o Open Source é um novo método de desenvolvimento de software, colaborativo, que denominou de **bazar**, pois é constituído de uma comunidade de desenvolvedores voluntários, que atuam no projeto sob suas próprias agendas e vontades.

O seu contraponto é o modelo tradicional, a **catedral**, em que o software é desenvolvido por uma equipe fechada e gerenciada de forma hierárquica, com cronogramas rígidos e os desenvolvedores atuam nas partes dos códigos que lhe foram designadas e não escolhem o que e quando desenvolver.

## 6. OUTROS CONCEITOS IMPORTANTES

### 6.1. ABSTRAÇÃO

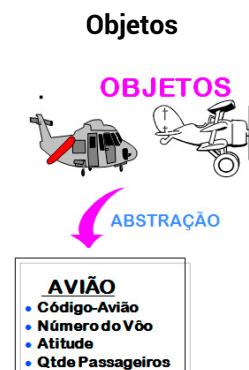
**Abstração** é o processo mental de separar um ou mais elementos de uma totalidade complexa, de forma a facilitar a sua compreensão. No nosso dia a dia utilizamos a abstração para poder trabalhar com toda a informação que o mundo nos fornece.

Rumbaugh destaca **abstração** como uma habilidade mental que permite aos seres humanos visualizarem os problemas do mundo real com vários graus de detalhe, dependendo do contexto corrente do problema.

Em outras palavras, **abstração** é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.

Em uma modelagem orientada a objetos, cabe destacar que uma classe é uma abstração de entidades existentes no domínio do sistema de software. Um objeto é uma ocorrência específica (instância) de uma classe.

Como exemplo, imagine a abstração referente à classe **Animais**. Há várias entidades na classe Animais como Anfíbios, Répteis e Mamíferos que são também subclasses da classe Animais, onde há **objetos** que contêm cada subclasse como Ser humano, Jacaré e outros.

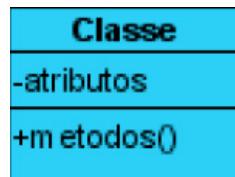


- Representam elementos do mundo real. É uma **abstração** de conjunto de coisas do mundo real.

- Possuem:
- **Atributos (estado);**
- **Operações (comportamento).**
- O único acesso aos dados desse objeto é por meio de suas **operações.**

## Classes

- Permitem que sejam representados no mundo computacional elementos do mundo real, ou seja, do problema para o qual o software está sendo desenvolvido.



- Elas permitem descrever um **conjunto de objetos** que compartilhem os mesmos atributos, operações, relacionamentos e semântica, e representam o principal bloco de construção de um software orientado a objetos.
- Representam os tipos de objetos existentes no modelo, e são descritas a partir de seus **atributos, métodos e restrições.**

A figura seguinte apresenta a simbologia para uma **CLASSE** chamada ContaBancaria, utilizando a linguagem UML.



Figura. Classe ContaBancaria

Com as classes definidas, precisam-se especificar quais são seus **ATRIBUTOS (propriedades que caracterizam um objeto)**. Por exemplo, uma entidade conta bancária possui como atributos o número e o saldo. É bastante simples identificar os atributos de cada classe, basta identificar as **características que descrevam sua classe no domínio do problema em questão**. Cabe destacar que os atributos identificados devem estar alinhados com as necessidades do usuário para o problema.

A figura seguinte apresenta a classe ContaBancaria com alguns de seus atributos.

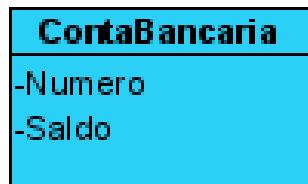


Figura. Classe ContaBancaria com alguns atributos

Identificadas as classes e seus atributos, o próximo passo é a **identificação das OPERAÇÕES de cada classe**, também chamadas de métodos ou serviços.

Fazendo um paralelo com objetos do mundo real, operações são **ações que o objeto é capaz de efetuar**. Dessa forma, ao procurar por operações, devem-se identificar ações que o objeto de uma classe é responsável por desempenhar dentro do escopo do sistema que será desenvolvido.

A figura seguinte apresenta algumas operações da classe ContaBancaria. Ao contrário dos atributos, normalmente operações são públicas, permitindo sua utilização por outras classes e objetos.

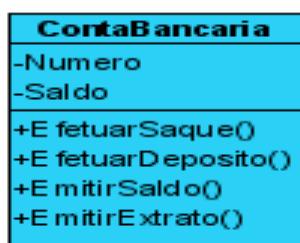


Figura. Classe ContaBancaria com seus atributos e operações

## 6.2. MODELAGEM

Como visto, um modelo é uma **representação abstrata de algo real**. Ele tem o objetivo de imitar a realidade, para possibilitar que esta seja estudada quanto ao seu comportamento.

**Os modelos permitem focalizar a atenção nas características importantes do sistema**, dando menos atenção às coisas menos importantes. Seu uso torna o estudo mais barato e seguro: é muito mais rápido e barato construir um modelo do que construir a coisa “real”.

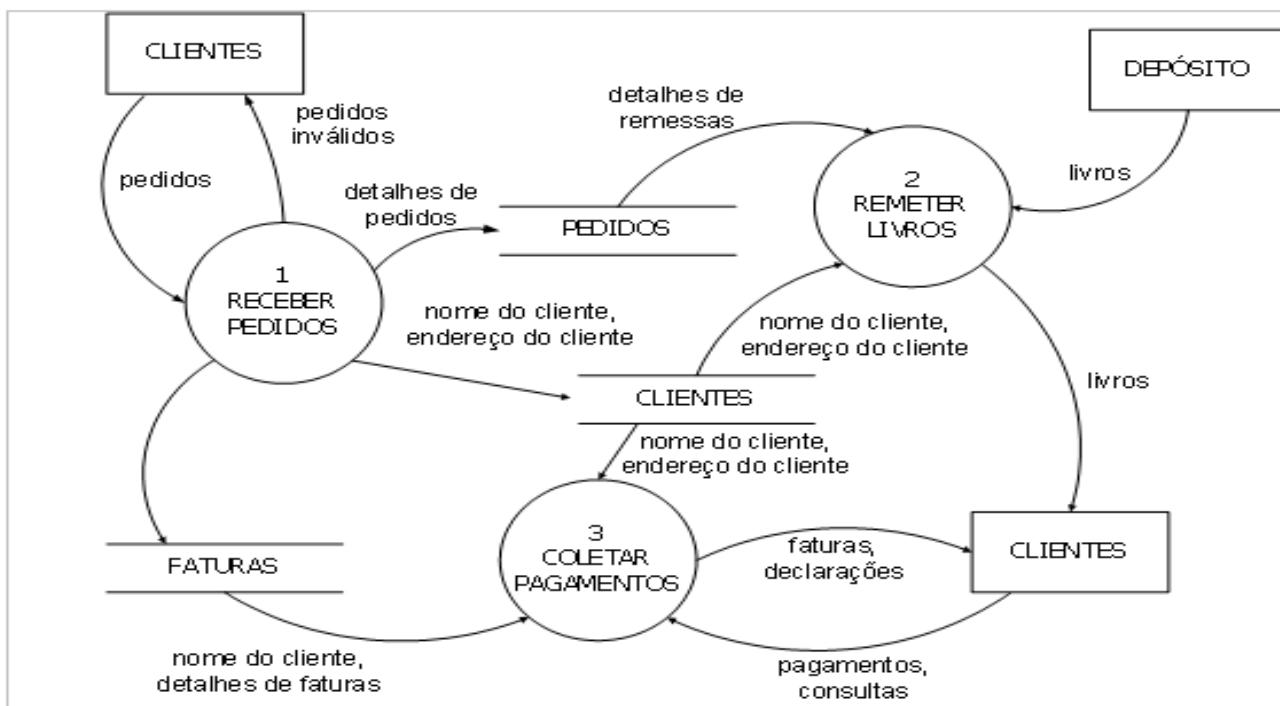
O objetivo do modelo é mostrar como será o sistema, para permitir sua inspeção e verificação, a fim de poder receber alterações e adaptações antes de ficar pronto. **Qualquer modelo realça certos aspectos do que está sendo modelado, em detrimento dos outros.**

## 7. CONCEITOS PRINCIPAIS RELACIONADOS À ANÁLISE ESTRUTURADA

- A **análise estruturada** é uma metodologia para desenvolvimento de sistemas que surgiu em meados da década de 70.
- Seu principal conceito é a **construção de um modelo lógico** (não físico) de um sistema, utilizando-se de gráficos capazes de levar usuários e analistas a formarem um quadro claro e geral do sistema e de como as suas partes se encaixam para atender às necessidades daqueles que dele precisam.
- O trabalho realizado com análise estruturada é também denominado Modelagem Funcional.
  - Este tipo de modelagem tem como objetivo definir **O QUE** o sistema deve fazer.

- A modelagem faz uso de técnicas que exigem que a análise de um sistema seja representada por **agentes externos, processos (ou funções), comunicações entre esses processos (fluxos de dados) e os dados necessários para que os processos façam seu trabalho (depósitos de dados)**.
- Essa metodologia envolve a construção de um sistema de forma
- **top-down** (do geral para o particular), por meio de **refinamentos sucessivos**, produzindo primeiro um fluxo de dados global do sistema, para depois desenvolver fluxos detalhados. São então detalhadas as estruturas de dados e a lógica dos processos.
- A análise estruturada usa linguagens (gráficas ou formais) que têm o poder de restringir as interpretações possíveis do que queremos dizer e que constituem as ferramentas usadas. Entre essas destacamos o **Diagrama de Fluxo de Dados (DFD)**, o **Dicionário de Dados (DD)** e o **Diagrama de Transição de Estados (DTE)**.
  - O **DFD** é uma representação em rede dos processos (funções) do sistema e dos dados que ligam esses processos. Ele mostra “o que” o sistema faz e não “como” é feito. É a ferramenta central da análise estruturada, descrevendo o modelo funcional.

Um DFD apresenta as partes componentes de um sistema e as interfaces entre elas, formando um conjunto integrado de procedimentos.



Na elaboração de um DFD utilizaremos **quatro símbolos** que nos permitirão debater e apresentar ao usuário todo o processo, sem assumir nenhum compromisso com implementações e sem a preocupação com a hierarquização e tomadas de decisão:



- O **dicionário de dados** é um repositório de dados sobre os dados do software. Ele deve fornecer informações sobre a definição, a estrutura e a utilização de cada elemento de dados que o sistema utiliza. Um Elemento de Dado é uma unidade de dados que não pode ser decomposta.
- **Diagrama de Transição de Estados (DTE)**: representa o comportamento de um sistema, descrevendo seus estados e os eventos que fazem com que o sistema mude de estado, devendo apresentar um único estado inicial e **0 ou vários estados finais!**

## 8. O PARADIGMA DA ORIENTAÇÃO A OBJETOS

A expressão “*orientado a objetos*” significa que o software é organizado como uma coleção de **objetos** separados que incorporam tanto a **estrutura** quanto o **comportamento** dos componentes do sistema.

Isto é diferente da **programação convencional**, em que a estrutura e o comportamento dos dados têm **poucos** vínculos entre si.

## 9. ASPECTOS DA ORIENTAÇÃO A OBJETOS

A orientação a objetos possui diferentes aspectos que a definem, as quais podem variar de acordo com o autor.

### 9.1. CLASSIFICAÇÃO

**Classificação** significa que os **objetos** com a mesma estrutura de dados (chamados **atributos**) e o mesmo **comportamento** (chamados **operações** ou **métodos**) são agrupados em uma **classe**. **Classe** é uma **abstração** que descreve propriedades importantes para uma aplicação e ignora o restante.

A escolha de classes depende da aplicação. Cada **classe** descreve um conjunto possivelmente infinito de objetos individuais e cada **objeto** é uma instância de sua classe.

Uma instância da classe tem seu próprio valor para cada atributo, mas compartilha os nomes de atributos e operações com outras instâncias da mesma classe.

## 9.2. IDENTIDADE

**Identidade** é a subdivisão dos dados em entidades discretas e distintas, que são objetos. Cada objeto tem sua própria identidade, ou seja, são distintos mesmo que todos os valores de seus atributos sejam idênticos, como nome e tamanho, por exemplo.

No mundo real, um objeto limita-se a existir, mas no que se refere à linguagem de programação, **cada objeto possui um ÚNICO indicador**, pelo qual ele pode ser referenciado sem erros.

## 9.3. ÊNFASE NA ESTRUTURA DE OBJETOS

O desenvolvimento baseado em **objetos** dá maior ênfase na estrutura de dados (em comparação com a estrutura de procedimentos) do que as metodologias tradicionais de decomposição funcional.

O desenvolvimento baseado em objetos é similar às técnicas de modelagem de informações utilizadas no projeto de bancos de dados, acrescentando o conceito de comportamento dependente da classe.

## 9.4. COMPARTILHAMENTO

O **compartilhamento (herança)** da estrutura de dados e do seu comportamento permite que a estrutura comum seja compartilhada por diversas subclasses semelhantes, **sem redundâncias**. O desenvolvimento baseado em objetos não somente permite que as informações sejam compartilhadas como também oferece a possibilidade da **reutilização** de modelos e códigos em projetos futuros.

Uma das principais vantagens das linguagens baseadas em objetos é o compartilhamento de código com utilização de herança, a qual reduz o trabalho de codificação e mostra o conceito proveniente de que as diferentes operações são, na realidade, as mesmas.

## 9.5. POLIMORFISMO

É quando a MESMA operação pode atuar de modos diversos em classes diferentes. Uma operação é uma ação (ou transformação) que um objeto executa (ou a que ele está sujeito). Uma implementação específica de uma operação por uma determinada classe é chamada de **método**.

Pode haver mais de um método para a implementação de um operador baseado em objetos, pois ele é polimórfico.

## 9.6. ENCAPSULAMENTO

O **encapsulamento**, também conhecido como **ocultamento de informações**, **consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, dos detalhes internos da implementação daquele objeto**.

O encapsulamento impede que um programa se torne tão dependente que uma pequena modificação possa causar grandes efeitos de propagação.

## 10. MODELAGEM ORIENTADA A OBJETOS

A **UML** (*Unified Modeling Language – Linguagem Unificada para Modelagem*) é a linguagem mais utilizada atualmente para especificação e projeto de software na abordagem orientada a objetos. A UML é o instrumento que permite a modelagem do software “visualmente”, tornando fácil partir dos requisitos do sistema à implementação de uma forma amigável.

## 11. ANÁLISE ORIENTADA A OBJETOS X ANÁLISE ESTRUTURADA

O objetivo da **Análise Orientada a Objetos** é desenvolver uma série de modelos que descrevem como o software irá se “comportar” para satisfazer seus requisitos. Logo, **a preocupação maior desta fase é representar “O QUE” o sistema irá fazer, sem considerar “como”**.

Como na **Análise Estruturada**, constroem-se modelos que representam diversas perspectivas, descrevendo o fluxo de informações, as funções e o comportamento do sistema dentro do contexto dos elementos do sistema. No entanto, **diferentemente da Análise Estruturada, aplica-se aqui os conceitos do paradigma da Orientação a Objetos para realizar-se o estudo do sistema, buscando-se um desenvolvimento mais adequado**.

Seguindo os modelos do processo de software, a Análise Orientada a Objetos é realizada a partir dos documentos de **Especificação de Requisitos** do software, os quais descrevem, principalmente, as funcionalidades esperadas do sistema.

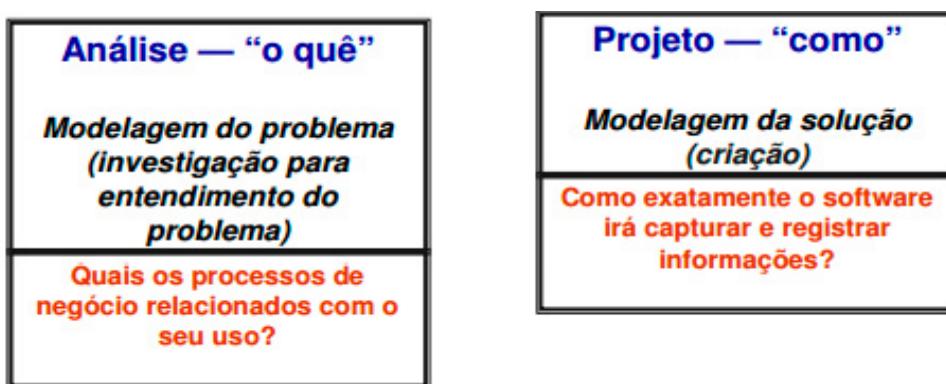
Nesta fase produz-se um conjunto de diagramas que descrevem as características do software. Deve-se ressaltar que o emprego dos diagramas difere de acordo com as características do sistema. Pode-se ainda desenvolver um Dicionário de Dados descrevendo os componentes dos diagramas. Ao final desta fase, se possível, deve-se revisar os documentos juntamente com representantes do cliente.

Os diagramas produzidos nesta fase são modelos gráficos representativos do sistema. Os diagramas gerados podem (e devem) ser empregados para a prototipação do sistema, podendo-se avaliar se ele cumpre realmente seus requisitos. **Na Análise Orientada a Objetos, podem-se empregar diferentes diagramas da UML, de acordo com a necessidade do projeto.**

## RESUMO

### Engenharia de Software

- É uma área da computação que visa abordar de modo sistemático as **questões técnicas e não técnicas** no **projeto, implantação, operação e manutenção** no desenvolvimento de um software;
- É uma disciplina que se **ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema**, depois que ele entrou em operação. Seu principal objetivo é fornecer uma estrutura metodológica para a construção de software com alta qualidade.
- De acordo com Pressman, a **engenharia de software** é baseada em camadas, com foco na qualidade. Essas camadas são: **ferramentas, métodos e processo**.
- Os princípios de engenharia de software definem a necessidade de **formalidades** para reduzir inconsistências e a **decomposição** para lidar com a complexidade.
- O *design* de software, ao descrever os diversos aspectos que estarão presentes no sistema quando construído, permite que se faça a avaliação prévia para garantir que ele alcance os objetivos propostos pelos interessados.



- A representação de um *design* de software mais simples para representar apenas as suas características essenciais busca atender ao princípio da **abstração**.
- Iniciar a entrevista para obtenção dos requisitos de software com perguntas mais genéricas e finalizar com perguntas mais específicas sobre o sistema é o que caracteriza a técnica de **entrevista estruturada em funil**.
- No contexto de levantamento de requisitos, **funcionalidade** é um dos aspectos que deve ser levado em conta na abordagem dos requisitos funcionais.
- A **engenharia de software** está inserida no contexto: das engenharias de sistemas, de processo e de produto.
- O **Ciclo de Vida de Software** trata das fases pelas quais um software passa desde o seu início até o seu fim.

- O **Modelo de Ciclo de Vida de Software** trata das fases pelas quais um software passa desde o seu início até o seu fim e como essas fases se relacionam (processo).
- O modelo de processo de desenvolvimento de software **incremental** que enfatiza um **ciclo de desenvolvimento extremamente curto**, que compreende as fases de modelagem do negócio, modelagem dos dados, modelagem do processo, geração da aplicação, além de teste e entrega, e que o desenvolvimento é conseguido pelo uso de construção baseada em componentes, é conhecido como modelo: **RAD**.
- **RAD** é um processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento curto.

## QUESTÕES COMENTADAS EM AULA

**001.** (CESPE/2015/MEC/ANALISTA DE SISTEMAS) Julgue o item a seguir, a respeito da engenharia de software.

Na fase de engenharia de requisitos do software, do paradigma do ciclo de vida clássico da engenharia de software chamado de modelo cascata, são identificadas as necessidades do sistema do ponto de vista do desenvolvedor, sem a presença do solicitante.

**002.** (CESPE/2016/FUNPRESP/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO) O modelo de execução de projetos em cascata é caracterizado por fases que se entrelaçam e se sobrepõem. A abordagem incremental, por sua vez, assemelha-se ao planejamento em ondas sucessivas.

**003.** (CESPE/TCE-RN/2009) A prototipação, uma abordagem para desenvolvimento de software na qual se cria um modelo do software que será implementado, é composta de quatro etapas: planejamento, análise de risco, engenharia e avaliação do cliente.

**004.** (FGV/2014/CM-RECIFE/ANALISTA LEGISLATIVO – ANALISTA DE SISTEMAS) Protótipos auxiliam na elicitação e na validação dos requisitos de sistemas computacionais. Duas das técnicas muito utilizadas durante uma prototipação são:

- a) prototipação em papel e etnografia;
- b) etnografia e wireframes;
- c) prototipação em papel e wireframes;
- d) mockups e pesquisa de mercado;
- e) entrevistas e pesquisa de mercado.

**005.** (CESPE/TRT – 10<sup>a</sup> REGIÃO (DF E TO)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO/2013) Tendo em vista que o desenvolvimento de um *software* comprehende várias fases, que vão desde a definição básica até o uso do *software*, e que, nesse processo, diversos modelos, métodos e procedimentos de construção podem ser utilizados, julgue os itens subsecutivos.

No ciclo de vida da primeira versão do modelo em espiral, a etapa de análise de riscos é realizada dentro da fase de desenvolvimento.

**006.** (CESPE/MPE-RN/GESTÃO E ANÁLISE/2010) O modelo em espiral difere principalmente dos outros modelos de processo de software por

- a) não contemplar o protótipo.
- b) reconhecer explicitamente o risco.
- c) não ter fases.

- d) possuir uma fase evolucionária.
- e) não contemplar o projeto do produto.

**007.** (CESPE/2013/MPOG/TECNOLOGIA DA INFORMAÇÃO) Ao realizar o TDD (test-driven development), o programador é conduzido a pensar em decisões de design antes de pensar em código de implementação, o que cria um maior acoplamento, uma vez que seu objetivo é pensar na lógica e nas responsabilidades de cada classe.

## QUESTÕES DE CONCURSO

**008.** (FUNRIO/2013/MPOG/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO/ENGENHARIA DE SOFTWARE) No processo unificado de desenvolvimento de software, qual é a fase em que o planejamento do projeto é completado, o domínio do negócio é analisado e os requisitos do sistema são ordenados considerando-se prioridade e risco?

- a) Concepção.
- b) Elaboração.
- c) Construção.
- d) Transição.
- e) Produção.



**Obs.:** O **Rational Unified Process (RUP)** é um exemplo de modelo de processo de desenvolvimento baseado no **Unified Process (Processo Unificado)** desenvolvido pela Rational.

O **Rational Unified Process** é um processo de desenvolvimento **iterativo** e **incremental**, no qual o *software* não é implementado em um instante no fim do projeto, mas é, ao contrário, desenvolvido e implementado em partes. A cada iteração deste processo utiliza-se **quatro** fases, a saber: **Concepção, Elaboração, Construção e Transição**.

Durante a **Concepção ou Iniciação (Inception)**, estabelece-se a lógica do domínio da aplicação para o projeto e decide o escopo do projeto. É aqui que se obtém o comprometimento do patrocinador do projeto para seguir adiante. Nesta fase comprehende-se o problema da tecnologia empregada por meio da definição dos *use cases* mais críticos. Define-se aqui o caso de negócios e escopo do projeto.

Na **Elaboração (Elaboration)** coleta-se requisitos mais detalhados, faz uma análise e um projeto de alto nível para estabelecer uma arquitetura básica, e cria um plano para a construção do sistema. Deve-se analisar o domínio do problema, estabelecer a arquitetura, desenvolver o plano do projeto e eliminar elementos de alto risco.

A fase de **Construção (Construction)** consiste de várias **iterações**, nas quais cada iteração constrói *software* de qualidade de produção, testado e integrado, que satisfaz um subconjunto de requisitos de projeto. Cada iteração contém todas as fases usuais do ciclo de vida da análise, do projeto, da implementação e do teste. Desenvolve-se o software e prepara-se o mesmo para a transição para os usuários. Além do código, também são produzidos os casos de teste e a documentação.

Mesmo com este tipo de processo iterativo, algum trabalho tem que ser deixado para ser feito no fim, na fase de **Transição (Transition)**. Nesta fase são realizados os treinamentos dos usuários e a transição do produto para utilização. Este trabalho pode incluir também testes beta e ajuste de performance.

**Letra b.**

**009.** (FGV/MPE-AL/ANALISTA DO MINISTÉRIO PÚBLICO/DESENVOLVIMENTO DE SISTEMAS/2018) O Processo Unificado de software é uma tentativa de aproveitar os melhores recursos e características dos modelos tradicionais de processo de software. Sobre o Processo Unificado de software, assinale a afirmativa correta.

- a) O software é dirigido a casos de uso, centrado na arquitetura, sequencial e incremental.
- b) O software é entregue aos usuários finais na fase de transição.
- c) Os modelos de caso de uso, análise, projeto e implementação são desenvolvidos na fase de concepção.
- d) O planejamento é realizado na fase de elaboração.
- e) Os requisitos não funcionais são descritos em um conjunto de casos de uso preliminares.



- a) **Errada.** O software é dirigido a casos de uso, centrado na arquitetura, iterativo e incremental.
- b) **Certa.** O software é entregue aos usuários finais na fase de **Transição (Transition)**. Nessa fase são realizados os treinamentos dos usuários e a transição do produto para utilização em um ambiente real.
- c) **Errada.** Os modelos de caso de uso, análise, projeto e implementação são desenvolvidos na **fase de elaboração**. Na **Elaboração (Elaboration)** requisitos mais detalhados são coletados, faz-se uma análise e um projeto de alto nível para estabelecer uma arquitetura básica, e é criado um plano para a construção do sistema. Deve-se analisar o domínio do problema, estabelecer a arquitetura, desenvolver o plano do projeto e eliminar elementos de alto risco.
- d) **Errada.** O planejamento é realizado na fase de **Concepção ou Iniciação (Inception)**.
- e) **Errada.** Os **requisitos funcionais** são descritos em um conjunto de casos de uso preliminares.

**Letra b.**

**010.** (FGV/ALERJ/ESPECIALISTA LEGISLATIVO – TECNOLOGIA DA INFORMAÇÃO/2017)

Um sistema está sendo desenvolvido com a utilização do processo unificado, que contém diversas fases. Na fase atual do processo será feita a implantação do sistema e a análise de lições aprendidas. Os analistas de requisitos e de negócio, praticamente, já terminaram suas atividades. É necessário ainda analisar a possibilidade de se executar outro ciclo de desenvolvimento. O sistema está na fase de:

- a) produção;
- b) concepção;
- c) elaboração;
- d) transição;
- e) construção.



O software é entregue aos usuários finais na fase de **Transição (Transition)**. Nessa fase são realizados os treinamentos dos usuários e a transição do produto para utilização em um ambiente real.

**Letra d.**

**011.** (FGV/SEE-PE/PROFESSOR DE DESENVOLVIMENTO DE SISTEMAS/2016) O sistema SISFORÇA está sendo desenvolvido com a utilização do processo unificado. Este processo contém diversas fases. Na fase atual do processo do SISFORÇA está sendo realizada a fusão de vários artefatos de software, possibilitando que o sistema seja implementado quase que completamente. Nessa fase, tem-se uma visão geral de como a Baseline do projeto está sendo seguida.

De acordo com o fragmento acima, o sistema SISFORÇA está na fase de

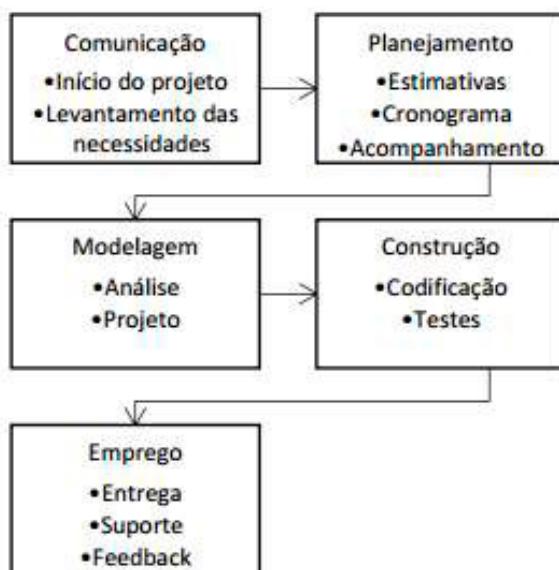
- a) produção.
- b) transição.
- c) elaboração.
- d) concepção.
- e) construção.



O **Processo Unificado de desenvolvimento de software** representa um conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de software. O cenário explicitado na questão destaca a fase de **Construção (Construction)**. Essa fase consiste de várias **iterações**, nas quais cada iteração constrói software de qualidade de produção, testado e integrado, que satisfaz um subconjunto de requisitos de projeto. Cada iteração contém todas as fases usuais do ciclo de vida da análise, do projeto, da implementação e do teste. Desenvolve-se o software e preparase o mesmo para a transição para os usuários. Além do código, também são produzidos os casos de teste e a documentação.

**Letra e.**

**012.** (FGV/COMPESA/ANALISTA DE GESTÃO/ADMINISTRADOR DE BANCO DE DADOS/2016) Observe a figura a seguir, que representa um modelo de processo de software.



Este modelo, algumas vezes chamado ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software nos casos em que os requisitos de um problema são bem compreendidos e quando o trabalho flui da comunicação ao emprego de forma relativamente linear. O modelo apresentado é denominado

- a) incremental.
- b) cascata.
- c) evolucionário.
- d) unificado
- e) especializado.



O **modelo em cascata** ou **ciclo de vida clássico**, segundo (Pressman, pg.32), “[...] requer uma abordagem sistemática, sequencial ao desenvolvimento de software, que se inicia no nível do sistema e avança ao longo da análise, projeto, codificação, teste e manutenção.”

**Letra b.**

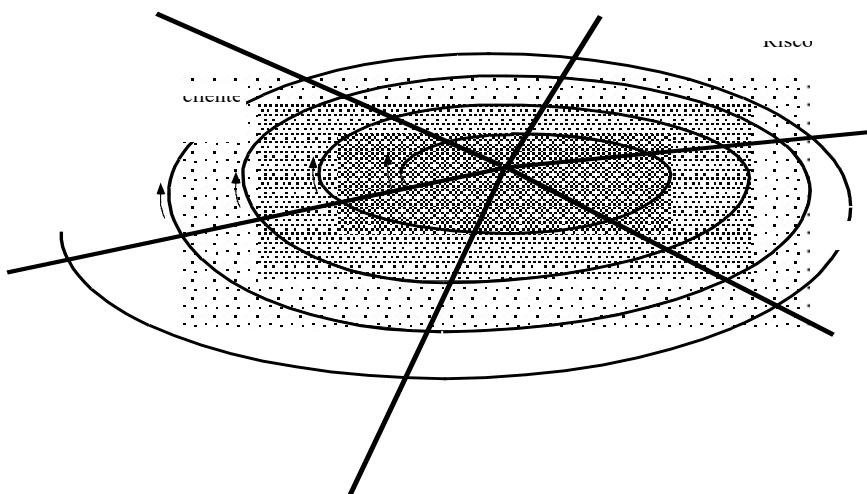
**013.** (FGV/IBGE/ANALISTA – ANÁLISE DE SISTEMAS/DESENVOLVIMENTOS DE SISTEMAS/2016) A empresa SONOVATOS desenvolve sistemas há pouco tempo no mercado e, como padrão, sempre utilizou o modelo Cascata de ciclo de vida. Alguns clientes ficaram insatisfeitos com os produtos desenvolvidos pela empresa por não estarem de acordo com suas necessidades. Atualmente a SONOVATOS está desenvolvendo sistemas muito maiores, com duração de vários anos, e com requisitos ainda instáveis. O próprio processo de desenvolvimento da empresa também está em reformulação.

Assim, a adoção de um novo modelo de ciclo de vida está sendo avaliada pelos gerentes da empresa. A intenção da SONOVATOS é, principalmente, gerenciar riscos e poder reavaliar constantemente o processo de desenvolvimento ao longo do projeto, o que permitiria correções nesse processo ou até mudança do tipo de processo. O modelo mais adequado para os sistemas atuais de longa duração da SONOVATOS é:

- a) Rapid Application Development (RAD);
- b) Espiral;
- c) Extremme Programming;
- d) Prototipação;
- e) Modelo V.



No **modelo em espiral** acrescenta-se a **Análise dos Riscos** ao ciclo de vida para auxiliar as decisões a respeito da próxima iteração. A figura seguinte apresenta o esquema deste modelo.



**Obs.:** Observe que cada *loop* na espiral representa uma fase do processo de software. Esse modelo exige a consideração direta dos riscos técnicos em todos os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que eles se tornem problemáticos.

**Letra b.**

**014. (FGV/DPE-RO/ANALISTA DA DEFENSORIA PÚBLICA – ANALISTA PROGRAMADOR/2015)** O Processo Unificado de desenvolvimento de software representa um conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de software. A fase do Processo Unificado destinada a endereçar os fatores de riscos conhecidos, e estabelecer e validar a arquitetura do software é a:

- a) Concepção;
- b) Elaboração;

- c) Construção;
- d) Transição;
- e) Produção.



A fase do Processo Unificado destinada a endereçar os fatores de riscos conhecidos, e estabelecer e validar a arquitetura do software é a: **Elaboração**. Algumas atividades dessa fase: coletam-se requisitos mais detalhados, faz-se uma análise e um projeto de alto nível para estabelecer uma arquitetura básica, e é criado um plano para a construção do sistema. Devese analisar o domínio do problema, estabelecer a arquitetura, desenvolver o plano do projeto e eliminar elementos de alto risco.

**Letra b.**

**015.** (FGV/TJ-PI/ANALISTA JUDICIÁRIO – ANALISTA DE SISTEMAS/BANCO DE DADOS/2015) Um sistema de informação está sendo desenvolvido com a utilização do Processo Unificado, que possui diversas fases. Na fase atual de desenvolvimento do sistema, os requisitos de negócio estão sendo refinados e será definida uma base de arquitetura executável. Essa arquitetura executável corresponde a uma evolução da arquitetura rudimentar que foi proposta na fase anterior. O sistema está atualmente na fase:

- a) Produção;
- b) Transição;
- c) Elaboração;
- d) Concepção;
- e) Construção.



A fase do Processo Unificado destinada a endereçar os fatores de riscos conhecidos, e estabelecer e validar a arquitetura do software é a: **Elaboração**. Algumas atividades dessa fase: coletam-se requisitos de negócio mais detalhados, faz-se uma análise e um projeto de alto nível para estabelecer uma arquitetura básica, e é criado um plano para a construção do sistema. Devese analisar o domínio do problema, estabelecer a arquitetura, desenvolver o plano do projeto e eliminar elementos de alto risco.

**Letra c.**

**016.** (FGV/PGE-RO/ANALISTA DA PROCURADORIA/ANALISTA DE SISTEMAS DESENVOLVIMENTO/2015) O Grupo de Estudo de Viabilidade de Projetos de uma empresa foi alocado para avaliar a informatização do procedimento de preservação digital de seus processos administrativos, e levantar as respostas para as seguintes questões:

O projeto é viável?

Qual é a ordem de grandeza dos custos, 10 ou 100 mil?

Devemos prosseguir com as próximas fases?

Aplicando o método Processo Unificado, essas questões deverão ser levantadas na fase de:

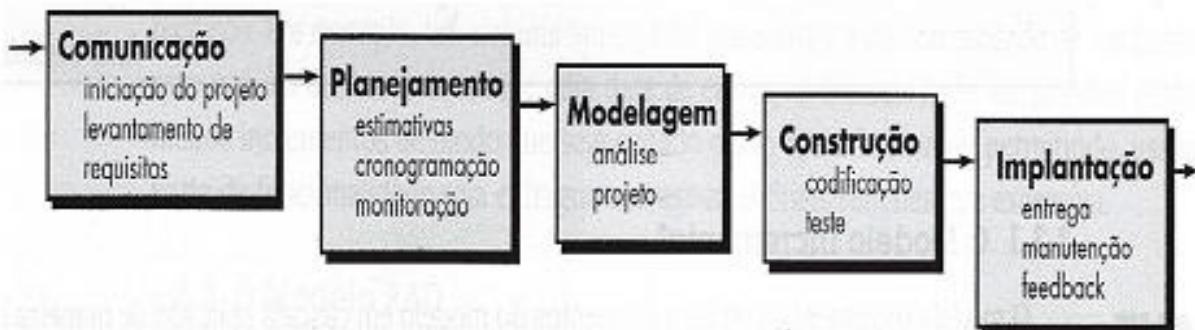
- Concepção;
- Elaboração;
- Construção;
- Transição;
- Análise.



O planejamento é realizado na fase de **Concepção ou Iniciação (Inception)** do Processo Unificado.

**Letra a.**

**017.** (FGV/FIOCRUZ/TECNOLOGISTA EM SAÚDE – TI/SISTEMAS DE INFORMAÇÃO / ENGENHARIA DE SOFTWARE/2010) A figura seguinte ilustra um modelo de processo, que prescreve um conjunto de elementos de processo como atividades de arcabouço, ações de engenharia de software, tarefas, produtos de trabalho, mecanismos de garantia de qualidade e de controle de modificações para cada projeto.



Esse modelo é conhecido como Modelo:

- por funções.
- em cascata.
- incremental.
- em pacotes.
- por módulos.



**Observe o uso das setas, simbolizando o término de uma fase e início de outra, o que caracteriza uma abordagem sequencial, típica do modelo em cascata. Também temos os nomes das fases do modelo, análise, projeto, codificação, teste e manutenção.**

A ideia principal do **Modelo em Cascata** ou **ciclo de vida clássico** é que **as diferentes etapas de desenvolvimento seguem uma sequência**: a saída da primeira etapa “flui” para a segunda etapa e a saída da segunda etapa “flui” para a terceira e assim por diante.

As atividades a executar são agrupadas em tarefas, executadas sequencialmente, de forma que uma tarefa só poderá ter início quando a anterior tiver terminado.

Após concluída uma etapa ou fase, deve ser feita uma homologação, a fim de verificar se os documentos gerados condizem com os requisitos do sistema. Sendo necessário, devem ser refeitas as tarefas com problemas.

Uma vez aprovada a fase, ela não deve ser alterada, pois implica em alteração na fase seguinte, e consequentemente aumento dos custos.

**Letra b.**

---

**018.** (CESPE/EBSERH/TÉCNICO EM INFORMÁTICA/2018) Em relação aos conceitos de engenharia de software, julgue o item subsecutivo. A engenharia de software se concentra nos aspectos práticos da produção de um sistema de software.

SOMMERVILLE (2011) destaca que a **engenharia de software** é uma **disciplina de engenharia que se preocupa com todos os aspectos da produção de software**, desde os estágios iniciais da especificação do sistema até a manutenção do sistema após sua utilização.



**Obs.:** Engenharia de software, segundo o IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), é a **aplicação de uma abordagem sistemática, quantificável e disciplinada para o desenvolvimento, operação e manutenção de software**.

Para Pressman, a engenharia de software é a **criação e utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais**.

Certo.

---

**019.** (CESPE/SLU-DF/ANALISTA DE GESTÃO DE RESÍDUOS SÓLIDOS – INFORMÁTICA/2019)

Acerca de conceitos e disciplinas da engenharia de software, julgue o item que se segue.

O processo de desenvolvimento de software, independentemente do seu tamanho e da sua destinação, pode envolver atividades genéricas como comunicação, planejamento, modelagem, construção e uso.



Podemos utilizar uma infinidade de modelos de processos, e a banca fez referência nessa questão ao modelo do Pressman (2011). Esse autor destaca que uma metodologia de processo genérica para Engenharia de Software compreende **cinco** atividades:

- **Comunicação** – Antes de iniciar o trabalho, é de vital importância comunicar-se e colaborar com o cliente (e outros interessados, como: executivos, usuários finais, engenheiros de software, o pessoal de suporte etc.). A intenção aqui é **compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades** que ajudarão a definir as funções e características do software.
- **Planejamento** – Estabelecimento do **plano de projeto de software**, que descreve as **tarefas técnicas** a ser conduzidas, os **riscos** prováveis, os **recursos** que serão necessários, os **produtos** resultantes a ser produzidos e um **cronograma** de trabalho.
- **Modelagem** – Criação de **modelos representativos do software** (para melhor entender as necessidades do software e o projeto que irá atender a essas necessidades).
- **Construção** – Essa atividade combina **geração de código** (manual ou automatizada) e **testes** necessários para revelar erros na codificação.
- **Entrega (Implantação, ou Uso)** – **Entrega do produto ao cliente**, que irá avaliá-lo e fornecer **feedback** baseado na avaliação.

Certo.

---

**020. (FCC/2014/TRT – 13ª REGIÃO (PB)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO)**

Flávio pretende desenvolver um *software* seguindo os estágios do modelo em cascata proposto por Sommerville, em razão de ponderações que faz em relação a outros modelos quanto à solução de um problema que se apresenta. Desta forma ele definiu em seu cronograma, na ordem apresentada pelo autor, as seguintes etapas do ciclo de vida de *software*:

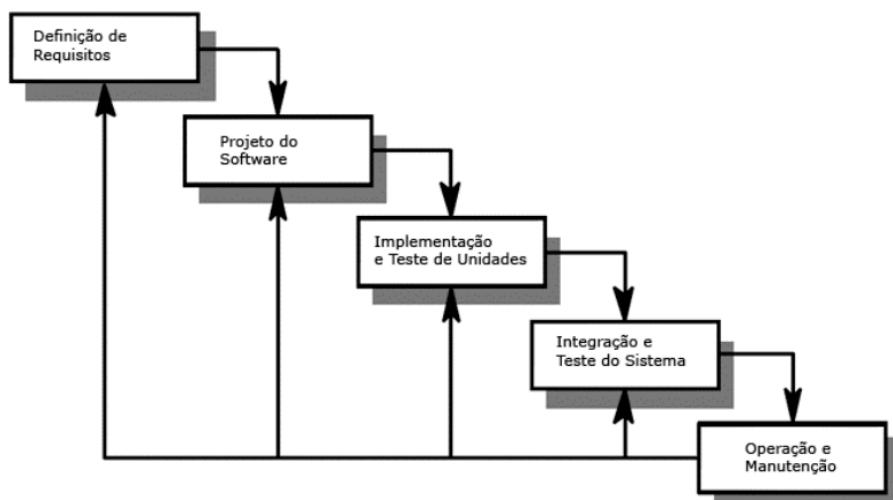
- a) Projeto de sistema e *software*; Definição de requisitos; Implementação e teste de unidade; Integração e teste de sistema; Operação e manutenção.
- b) Projeto de sistema e *software*; Engenharia de *software*; Integração e teste de sistema; Análise de requisitos funcionais e técnicos; Operação e manutenção; Implementação e teste de unidade.
- c) Projeto de sistema e *software*; Análise de requisitos; Engenharia de requisitos; Implantação; Testes de sistemas; Operação e manutenção.
- d) Definição de requisitos; Engenharia de requisitos; Integração e teste de sistema; Projeto de sistema e *software*; Implementação e teste de unidade; Operação e manutenção; Integração e teste de sistema.
- e) Definição de requisitos; Projeto de sistema e *software*; Implementação e teste de unidade; Integração e teste de sistema; Operação e manutenção.



De acordo com Sommerville, o **modelo em Cascata** possui as seguintes fases:

- **Análise e definição de requisitos:** nessa fase as funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta a representantes dos clientes e usuários do sistema;
- **Projeto de sistemas e de software:** define-se a arquitetura do sistema geral;
- **Implementação e teste de unidades:** nessa fase o projeto de software é desenvolvido e são realizados os testes de unidade para verificar que cada parte do software atenda a sua especificação;
- **Integração e teste de sistemas:** as partes (ou unidades) do programa são integrados e testados, formando um sistema completo. Após os testes, o software é liberado para o cliente; e
- **Operação e Manutenção:** aqui o software é instalado e colocado em operação.

A figura a seguir apresenta este modelo.



Referência: <http://marcosmoraidesousa.blogspot.com.br/2012/04/engenharia-de-software.html> -

**Letra e.**

**021.** (FCC/2014/TRT – 16<sup>a</sup> REGIÃO (MA)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO/ENGENHARIA DE SOFTWARE) Os modelos de processo são uma representação abstrata de um processo de software, que podem ser usados para explicar diferentes abordagens para o desenvolvimento de sistemas. Analise as seguintes abordagens:

- I – Desenvolvimento intercala as atividades de especificação, desenvolvimento e validação. Um sistema inicial é desenvolvido rapidamente baseado em especificações abstratas e depois é refinado com as entradas do cliente para produzir um produto que o satisfaça.
- II – Modelo considera as atividades fundamentais do processo, compreendendo especificação, desenvolvimento, validação e evolução e as representa como fases de processo separadas, tais como especificação de requisitos, projeto de software, implementação, teste etc.
- III – Baseia-se na existência de um número significativo de partes reusáveis. O processo de desenvolvimento do sistema enfoca a integração destas partes, ao invés de desenvolvê-las a partir do zero.

Os modelos de processo genéricos descritos em I, II e III são, correta e respectivamente, associados a:

- a) em Espiral – Baseado em Componentes – RAD
- b) Evolucionário – em Cascata – Baseado em Componentes
- c) Baseado em Componentes – Sequencial – Refactoring
- d) Ágil – Sequencial – Unified Process
- e) em Cascata – Ágil – Refactoring



Relembrando os conceitos de modelos de processo:

O modelo em **Cascata** (Waterfall), também chamado de Clássico, é o mais tradicional processo de desenvolvimento de software. Este modelo sugere uma abordagem seqüencial para o desenvolvimento de software, aplicando as atividades de maneira linear.

Já no Modelo **Incremental (Evolucionário ou Evolutivo)**, os primeiros incrementos são versões simplificadas do produto final, mas oferecem capacidades que servem ao usuário, além de servir como uma plataforma de avaliação. Neste modelo, em cada iteração uma parte é concebida como a menor unidade que pode ser implementada e ainda assim fornecer alguma funcionalidade útil para os usuários.

Na **Prototipagem (Prototipação)** os objetivos do software são estabelecidos na comunicação com o cliente. A partir daí, um protótipo e novas funcionalidades são acrescentadas de acordo com as demandas dos usuários.

No modelo **Espiral**, assume-se que o processo de desenvolvimento ocorre em ciclos, cada um contendo fases de avaliação e planejamento onde a opção de abordagem para a próxima fase (ou ciclo) é determinada. Este modelo foi proposto por Boehm como forma de integrar os diversos modelos existentes, eliminando suas dificuldades e explorando seus pontos fortes. Ele combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo cascata.

A Engenharia de Sistemas Baseada em **Componentes** (CBSE) a princípio parece bastante semelhante à engenharia de software convencional ou orientada a objetos. O processo começa quando uma equipe de software estabelece os requisitos para um sistema a ser construído usando técnicas de coleta de requisitos convencionais. Um projeto arquitetural é estabelecido, mas ao invés de passar imediatamente a tarefas de projeto mais detalhadas, a equipe examina os requisitos para determinar que subconjunto é mais adequado à composição, do que à construção.

Logo, a primeira afirmativa refere-se ao modelo Evolucionário, a segunda ao modelo Cascata e a terceira ao modelo Baseado em Componentes.

**Letra b.**

**022.** (CESPE/2013/TRT – 10ª REGIÃO (DF E TO)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO) Com relação a conceitos gerais de engenharia de *software*, julgue os itens a seguir.

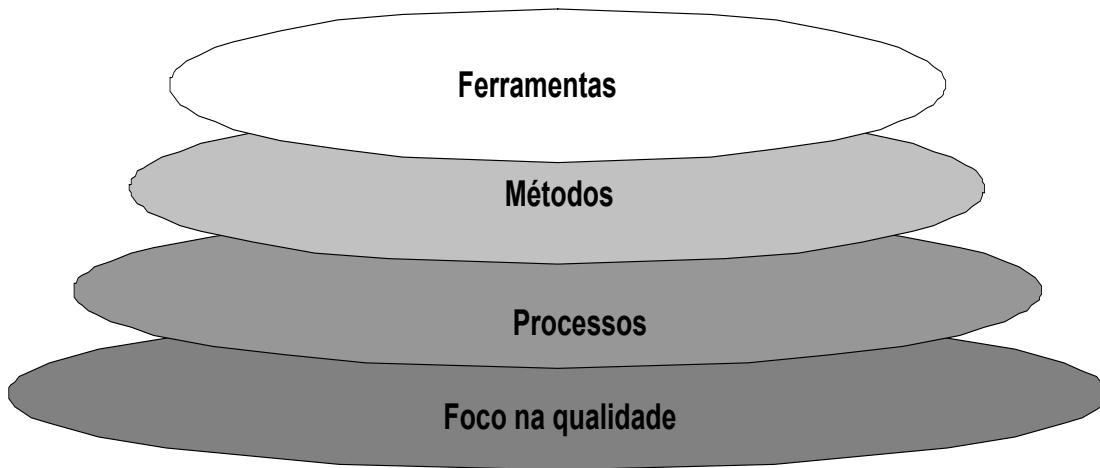
A engenharia de *software* engloba processos, métodos e ferramentas. Um de seus focos é a produção de *software* de alta qualidade a custos adequados.



- **Processo de Software:** é um conjunto de atividades, cuja meta é o desenvolvimento ou a evolução do software.
- **Métodos de Engenharia de Software:** são as abordagens estruturadas para o desenvolvimento de software, que incluem modelos de sistemas, notações, regras, recomendações de projetos e diretrizes de processos.

Apoiados pelos processos, **os métodos indicam “como fazer” para construir um software**. Para a realização dos métodos são aplicadas **ferramentas**, cujo objetivo é automatizar ou semi-automatizar os mesmos.

Logo, define-se a Engenharia de Software como uma tecnologia em camadas, conforme apresentado na próxima figura.



A utilização da Engenharia de Software tem como principais objetivos:

- Reducir o custo e o tempo de desenvolvimento;
- Possibilitar uma melhor gerência do processo de desenvolvimento;
- Facilitar o trabalho em grupo; e
- Aumentar a qualidade do produto.

**Certo.**

**023.** (FCC/2013/DPE-SP/AGENTE DE DEFENSORIA – PROGRAMADOR) No desenvolvimento de software podem ser utilizados diversos tipos de processo de desenvolvimento, dentre eles, processos iterativos. Sobre o desenvolvimento iterativo de software é correto afirmar:

- a)** É adequado para aplicações bem compreendidas, com saídas previsíveis desde a análise e projeto e que não apresentem incertezas substanciais em seus requisitos.
- b)** Na primeira iteração, desenvolve-se uma parte do software, que deve ser utilizado e avaliado. Em seguida, incorpora-se o que se aprendeu na iteração anterior e se repete o ciclo. No final, assim que o cliente testou a parte do **software** que é apenas uma prova conceito, o código é intencionalmente abandonado, pois o produto final será entregue em seu lugar.
- c)** Os desenvolvedores precisam integrar todas as versões de artefatos do sistema e verificar-las no final de uma iteração. Além disso, cada iteração deve produzir uma versão executável do software.
- d)** O desenvolvimento iterativo consiste em uma série de iterações com duração máxima de um mês e com, no máximo, seis iterações a cada três meses.
- e)** Os desenvolvedores realizam as fases do processo de desenvolvimento de software em uma sequência linear rígida, onde cada etapa deve ser concluída antes que a seguinte comece.



**Em um processo iterativo, assume-se que o desenvolvimento ocorre em ciclos.** Este tipo de processo é indicado principalmente para situações em que o cliente consegue definir apenas

um conjunto de objetivos gerais do software, não identificando claramente seus requisitos, contradizendo a alternativa A.

Além disso, não é necessário aplicar um processo iterativo somente para desenvolvimento de protótipos descartáveis, conforme afirma a alternativa B.

Ainda, não existe uma limitação de tempo e nem de quantidade de iterações. E nem mesmo as fases devem seguir uma sequência linear rígida. Logo as alternativas D e E estão incorretas.

Somente a afirmativa C está correta. **Ao final de cada iteração, as versões são integradas, produzindo-se versões estáveis do software.**

**Letra c.**

---

**024.** (FUMARC/BDMG/2011-07) O modelo de ciclo de vida de processo de software cujos principais subprocessos são executados em estrita sequência, o que permite demarcá-los como pontos de controle bem definidos, é denominado:

- a) Espiral.
- b) Cascata.
- c) Prototipagem evolutiva.
- d) Dirigidos por prazo.



O modelo de ciclo de vida de processo de software cujos principais subprocessos são executados em estrita sequência, o que permite demarcá-los como pontos de controle bem definidos, é denominado **Modelo em Cascata (Waterfall)**, também chamado de Clássico, que é o mais tradicional processo de desenvolvimento de software.

Este modelo sugere uma abordagem sequencial para o desenvolvimento de software, aplicando as atividades de maneira linear. Em cada fase desenvolvem-se artefatos (produtos de software) que servem de base para as fases seguintes.

O modelo em Cascata possui como vantagem principal a simplicidade para a sua aplicação e gerência. No entanto, algumas desvantagens podem ser observadas:

Projetos reais raramente seguem este fluxo sequencial.

Dificuldade do cliente em declarar todas as suas necessidades no início do projeto.

Demora em apresentar resultados ao cliente.

**Letra b.**

---

**025.** (FUNCAB/MP-RO/ANALISTA/2012) Suponha que seu cliente tenha solicitado o desenvolvimento de um novo software. O modelo mais adequado para o gerenciamento deste processo de desenvolvimento de software, levando em conta as informações abaixo, é:

- o cliente não possui uma visão clara de todos os requisitos da aplicação.
- o cliente quer avaliar a viabilidade de desenvolvimento da aplicação.

- o cliente alocará um usuário-chave no projeto, em tempo integral, a fim de que este possa participar ativamente de todas as fases do projeto.
- o cliente gostaria de ter uma versão preliminar do sistema, com base em uma versão inicial dos requisitos, ainda que isto demande um investimento inicial.

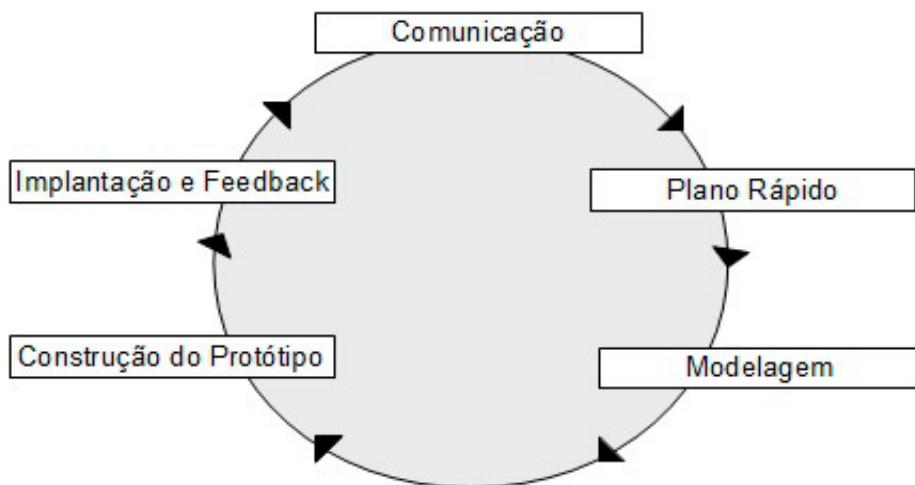
- a)** Cascata.  
**b)** Cíclico.  
**c)** Desenvolvimento Iterativo e Incremental.  
**d)** Prototipação.  
**e)** Formal.



Em algumas situações, o cliente consegue definir apenas um conjunto de objetivos gerais do software, não identificando claramente seus requisitos. Em uma situação dessas, a Prototipação (Prototipagem) pode ser empregada em conjunto com outros modelos para auxiliar no entendimento do sistema.

Os objetivos do software são estabelecidos na comunicação com o cliente. A partir daí, um protótipo descartável é elaborado com o intuito de facilitar a compreensão do sistema por parte dos usuários.

Apesar da prototipagem poder ser aplicada como um modelo, em geral ela é mais utilizada como uma técnica para entendimento do sistema. A próxima figura apresenta o esquema deste modelo.



#### Vantagens da prototipagem:

Maior participação e comprometimento dos clientes e usuários; e

Os resultados são apresentados mais rapidamente.

#### Críticas:

Forte dependência das linguagens e ambientes utilizados, bem como da experiência da equipe;

O cliente tende a considerar o protótipo como versão final, podendo comprometer a qualidade do projeto; e

O desenvolvedor tende a fazer concessões na implementação, a fim de colocar um protótipo em funcionamento rapidamente. Estas concessões podem se tornar parte integrante do sistema.

**Letra d.**

**026.** (CESGRANRIO/FUNASA/2009) À luz da Engenharia de Software, o ciclo de vida clássico, também chamado de modelo sequencial linear ou modelo em cascata, é um paradigma aplicável ao desenvolvimento de sistemas de informações que

- a)** enfatiza a análise de riscos, que é feita no início do projeto e revisada a cada fase, incluindo um plano de ataque e ações de mitigação dos riscos, aumentando as chances de sucesso do projeto.
- b)** prevê uma sequência (ou cascata) de entregas de versões do sistema ao longo do desenvolvimento do mesmo, o que proporciona uma avaliação de progresso baseada em código funcionando, em vez de quantidade de documentação gerada.
- c)** exige que todos os requisitos sejam conhecidos e corretamente especificados no início do ciclo de vida, dificultando a acomodação de mudanças que surjam nas fases posteriores.
- d)** foi sempre pouco utilizado na prática, pois se apoia em analogias com práticas de engenharia convencional que não se aplicam bem ao desenvolvimento de sistemas de informação.
- e)** utiliza a estratégia dividir para conquistar (divide to conquer), prevendo que cada fase do ciclo de vida seja desdobrada em um miniciclo de vida sequencial completo, formando uma cascata de ciclos de vida até o limite adequado para lidar com a complexidade do sistema.



O modelo em Cascata (Waterfall), também chamado de Clássico, é o mais tradicional processo de desenvolvimento de software. Este modelo sugere uma abordagem sequencial para o desenvolvimento de software, aplicando as atividades de maneira linear. Exige, portanto, que todos os requisitos sejam conhecidos e corretamente especificados no início do ciclo de vida, dificultando a acomodação de mudanças que surjam nas fases seguintes. Esse modelo possui como vantagem principal a simplicidade para a sua aplicação e gerência. No entanto, algumas desvantagens podem ser observadas:

- projetos reais raramente seguem este fluxo sequencial.
- dificuldade do cliente em declarar todas as suas necessidades no início do projeto, dificultando a implementação de mudanças que surjam nas fases posteriores;
- demora em apresentar resultados ao cliente.

**Letra c.**

**027. (ESAF/SEFAZ-CE/2007)** Analise a descrição a seguir.

O paradigma do ciclo de vida clássico da engenharia de software abrange seis atividades. Na atividade de \_\_\_\_\_ são traduzidas as exigências de uma representação do software que podem ser avaliadas quanto à qualidade antes que se inicie a codificação.

Escolha a opção que preenche corretamente a lacuna acima.

- a) análise
- b) engenharia de sistemas
- c) teste e análise de riscos
- d) coleta de requisitos
- e) projeto

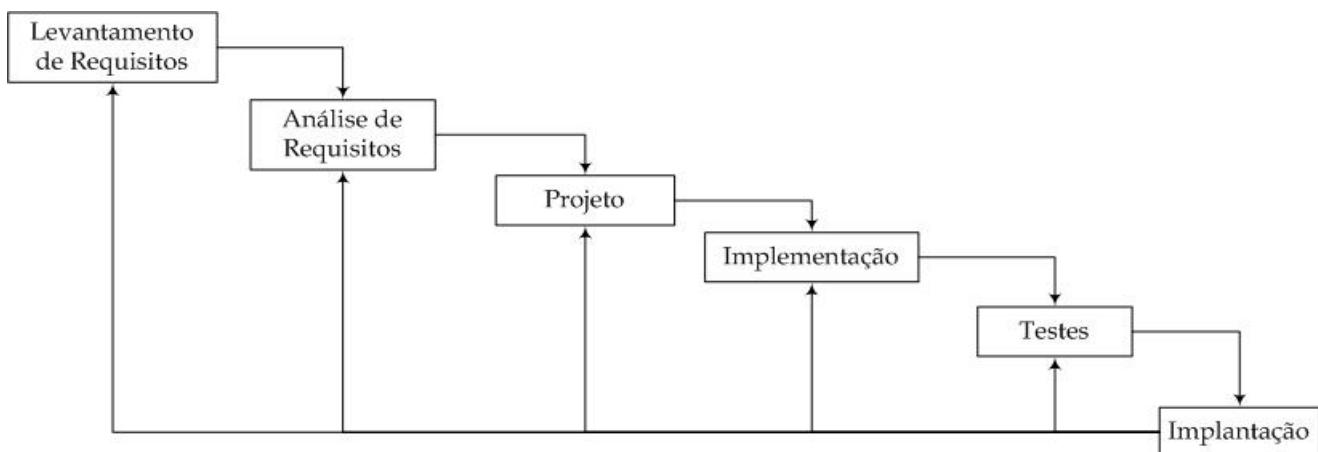


Como todo produto industrial, o produto de software tem seu **ciclo de vida**:

- ele é concebido para tentar atender a uma necessidade;
- é especificado, quando essas necessidades são traduzidas em requisitos viáveis;
- é desenvolvido, transformando-se em um conjunto formado por código e outros itens, como modelos, documentos e dados;
- passa por algum procedimento de aceitação e é entregue a um cliente;
- entra em operação, é usado, e sofre atividades de manutenção, quando necessário;
- é retirado de operação ao final de sua vida útil.

O ciclo de vida compreende muitas atividades, que são assunto das diferentes áreas da Engenharia de Software.

A figura seguinte mostra um **modelo do ciclo de vida clássico da engenharia de software**, que é dividido em **seis** atividades. São elas:



**Figura. Modelo em Cascata proposto por Bezerra.**

**Levantamento de Requisitos:** tem por objetivo propiciar que usuários e desenvolvedores tenham a mesma compreensão do problema a ser resolvido.

**Análise:** tem por objetivo construir modelos que determinam qual é o problema para o qual estamos tentando conceber uma solução de software.

**Projeto:** estágio no qual o modelo de análise terá de ser adaptado de tal modo que possa servir como base para implementação no ambiente alvo.

**Codificação (implementação):** a codificação do sistema é efetivamente executada.

**Teste:** consiste na verificação do software.

**Implantação:** entrada em produção do sistema.

Foi mencionado nessa questão que a atividade procurada é realizada antes da atividade de codificação, ou seja, antes da implementação do sistema.

A atividade que procuramos é responsável por descrever **COMO** o sistema será implementado, por meio da cobertura completa dos requisitos, traduzindo-os numa especificação (KRUCHTEN, 2003), e a resposta correta é Projeto!

**Letra e.**

**028. (ESAF/CGU/ ANALISTA DE FINANÇAS E CONTROLE/DESENVOLVIMENTO DE SISTEMAS DA INFORMAÇÃO/2012)** A escolha de um modelo é fortemente dependente das características do projeto. Os principais modelos de ciclo de vida podem ser agrupados em três categorias principais:

- a) sequenciais, cascata e evolutivos.
- b) sequenciais, incrementais e ágeis.
- c) sequenciais, incrementais e evolutivos.
- d) sequenciais, ágeis e cascata.
- e) cascata, ágeis e evolutivos.



Os principais modelos de ciclo de vida podem ser agrupados em três categorias principais que são: **sequenciais, incrementais e evolutivos**.

Modelos Sequenciais	Modelos Incrementais	Modelos Evolutivos
São aqueles em que o trabalho flui de forma relativamente linear (Exemplo: Modelo em Cascata).	Nos <b>modelos incrementais</b> , em cada iteração do ciclo acrescentam-se novas funcionalidades ao software (Exemplo: RAD).	No <b>modelo evolutivo</b> o software é desenvolvido em ciclos, e a cada ciclo novas funcionalidades são incrementadas ao sistema (Exemplo: Modelo Espiral, Prototipagem).

**Letra c.**

**029. (CESPE/2008/SERPRO/ANALISTA – DESENVOLVIMENTO DE SISTEMAS)**

Considerando os modelos do ciclo de vida de software, julgue o item que se segue.

O modelo em cascata consiste de fases e atividades que devem ser realizadas em sequência, de forma que uma atividade é requisito da outra.



O **modelo em Cascata (Waterfall)**, também chamado de **Clássico**, é o mais tradicional processo de desenvolvimento de software. Este modelo sugere uma abordagem sequencial para o desenvolvimento de software, aplicando as atividades de maneira linear. Em cada fase desenvolvem-se artefatos (produtos de software) que servem de base para as fases seguintes.

- Tendência na progressão sequencial entre uma fase e a seguinte.

**Certo.**

**030. (CESPE/2013/TRT – 10ª REGIÃO (DF E TO) – ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO)** Tendo em vista que o desenvolvimento de um *software* compreende várias fases, que vão desde a definição básica até o uso do *software*, e que, nesse processo, diversos modelos, métodos e procedimentos de construção podem ser utilizados, julgue os itens subsecutivos.

No modelo prototipação, a construção de *software* tem várias atividades que são executadas de forma sistemática e sequencial.



O **Modelo em Cascata** (Também chamado de clássico, ou linear) sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, aplicando as atividades de maneira linear. Em cada fase desenvolvem-se artefatos (produtos de software) que servem de base para as fases seguintes.

A prototipação consiste em confirmar ou refutar hipóteses. Senta-se com o cliente e realiza um projeto rápido para atender somente a aspectos do software que ficarão visíveis (protótipo). Para tal, o prazo de construção do mesmo deve ser obrigatoriamente curto, o que não possibilita formalismos (“abordagem sequencial e sistemática”), ao contrário do que foi sugerido no enunciado da questão.

**Errado.**

**031. (CESGRANRIO/PETROBRAS/PROCESSO DE NEGÓCIOS/2010)** Existem vários Modelos de Ciclo de Vida do Processo de Software. O desenvolvimento em espiral é um modelo

- a) iterativo.
- b) para modificar requisitos.
- c) para analisar componentes.
- d) para analisar interface gráfica.
- e) para modularizar o sistema.



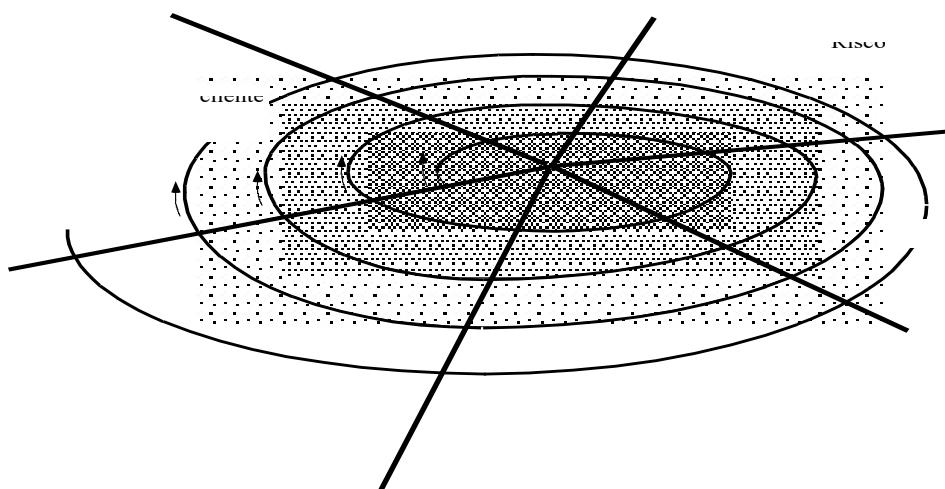
Um **modelo de ciclo de vida do software** deve conter a descrição precisa dos produtos de software gerados e dos critérios de término para cada fase, pois sem os mesmos o modelo em questão é considerado de pouca utilidade prática.

A escolha de um modelo de processo de software deve considerar as características do sistema, os métodos e as ferramentas a serem utilizados, os prazos e custos e ainda, os produtos de software a serem entregues. Alguns exemplos de modelos de ciclo de vida: Modelo em Cascata, Modelo Incremental, Modelo Espiral etc.

**O Modelo Espiral foi proposto por Boehm, em 1988, como forma de integrar os diversos modelos existentes, eliminando suas dificuldades e explorando seus pontos fortes.** Combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo cascata. Iterações sucessivas constroem um conjunto de artefatos a partir do estado em que estes foram deixados ao término da iteração passada.

Um projeto que usa o **desenvolvimento iterativo** tem um ciclo de vida que consiste em várias iterações. Uma iteração incorpora um conjunto quase sequencial de atividades em modelagem de negócios, requisitos, análise e projeto, implementação, teste e implantação, em várias proporções, dependendo do local em que ela está localizada no ciclo de desenvolvimento.

No modelo Espiral assume-se que o **processo de desenvolvimento ocorre em ciclos**, cada um contendo fases de avaliação e planejamento onde a opção de abordagem para a próxima fase (ou ciclo) é determinada. Neste modelo acrescenta-se a Análise dos Riscos ao ciclo de vida para auxiliar as decisões a respeito da próxima iteração. A próxima figura apresenta o esquema deste modelo.



Apesar desse modelo reunir as melhores características dos outros, algumas críticas podem ser feitas:

- Exige gerentes e técnicos experientes;
- Uma vez que o modelo em espiral pode levar ao desenvolvimento em paralelo de múltiplas partes do projeto, as tarefas gerenciais para acompanhamento e controle do projeto são mais complexas;
- É necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

**Letra a.**

**032.** (VUNESP/2013/CTA – TECNOLOGISTA PLENO – COMPUTAÇÃO/ENGENHARIA DE SOFTWARE) Considere o modelo espiral de ciclo de vida de software no qual as fases sejam:

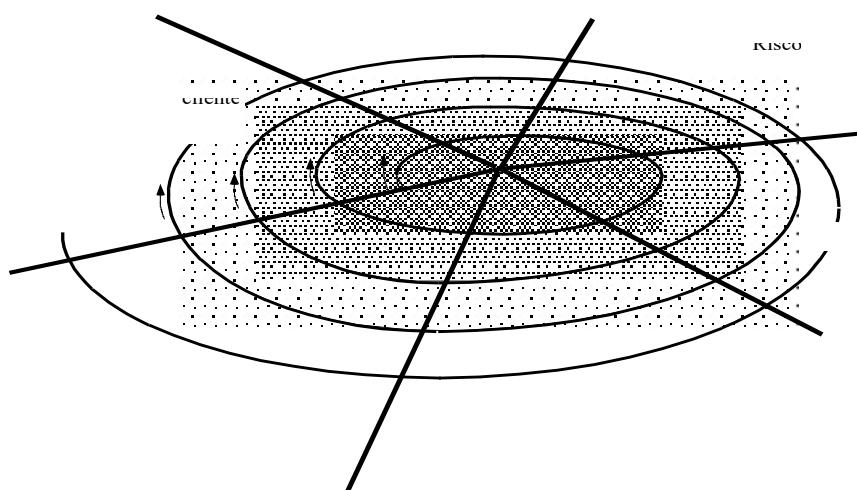
1. Modelagem,
2. Implantação,
3. Comunicação,
4. Planejamento,
5. Construção.

A ordem cronológica recomendada para a execução dessas 5 fases é:

- a) 2, 3, 1, 5, 4.
- b) 3, 1, 2, 5, 4.
- c) 3, 4, 1, 5, 2.
- d) 4, 1, 2, 5, 3.
- e) 4, 3, 2, 5, 1.



A próxima figura apresenta o esquema deste modelo.



Conforme visto, as fases são:

3. Comunicação.
4. Planejamento.
1. Modelagem.

**5. Construção.****2. Implantação.****Letra c.**

**033.** (CESPE/TRE-BA/ANALISTA DE SISTEMAS/2010) Com relação à engenharia de software, julgue o item a seguir.

Entre os desafios enfrentados pela engenharia de software estão lidar com sistemas legados, atender à crescente diversidade e atender às exigências quanto a prazos de entrega reduzidos.



A proposta da Engenharia de software é possibilitar o desenvolvimento adequado do software, frente a inúmeros desafios, como: lidar com sistemas legados (sistemas de software mais velhos que permanecem vital para uma organização), atender à crescente diversidade e atender às exigências quanto a prazos de entrega limitados.

**Certo.**

**034.** (FCC/2014/CÂMARA MUNICIPAL DE SÃO PAULO – SP/CONSULTOR TÉCNICO LEGISLATIVO/INFORMÁTICA) O desenvolvimento de uma solução para um sistema de informação baseia-se no processo de resolução de problemas. Esse processo pode ser descrito em quatro passos:

1. Definição e entendimento do problema.
2. Desenvolvimento de soluções alternativas.
3. Escolha da melhor solução.
4. Implementação da solução.

A seguir são descritas três atividades que ocorrem neste processo:

- I – Define cuidadosamente os objetivos do sistema modificado ou do novo sistema e desenvolve uma descrição detalhada das funções que um novo sistema deve desempenhar.
- II – Define se cada alternativa de solução é um bom investimento, se a tecnologia necessária para o sistema está disponível e pode ser administrada pela equipe designada da empresa, e se a organização é capaz de acomodar as mudanças introduzidas pelo sistema.
- III – É a “planta” ou modelo para a solução de um sistema de informação e consiste em todas as especificações que executarão as funções identificadas durante a análise de sistemas. Essas especificações devem abordar todos os componentes organizacionais, tecnológicos e humanos da solução.

A associação correta das atividades I, II e III aos passos ao qual pertencem no processo de resolução de problemas está, correta e respectivamente, apresentada em

**a) Gerenciamento de Requisitos – Passo 1**

Análise de Risco – Passo 3

Projeto de Sistema – Passo 3

**b) Análise de Requisitos – Passo 1**

Análise de Risco – Passo 3

Projeto de Sistema – Passo 4

**c) Elicitação de Requisitos – Passo 1**

Estudo de Viabilidade – Passo 2

Projeto de Sistema – Passo 4

**d) Gerenciamento de Requisitos – Passo 1**

Análise de Risco – Passo 2

Projeto de Sistema – Passo 3

**e) Análise de Requisitos – Passo 1**

Estudo de Viabilidade – Passo 3

Projeto de Sistema – Passo 4



A primeira afirmativa refere-se à **análise de requisitos**, que é definida como o uso sistemático de princípios, técnicas, linguagens e ferramentas comprovadas para análise, documentação, evolução continuada das necessidades dos usuários e especificação do comportamento externo de um sistema para satisfazer as necessidades do usuário, que sejam efetivas em termos de custos. Ela busca, principalmente, o entendimento escrito do problema.

Já a segunda afirmativa destaca o **estudo de viabilidade**. Nesse contexto, define-se, dentre as alternativas possíveis, qual solução será empregada, avaliando-se sob aspectos tecnológicos e organizacionais se o projeto é viável.

Finalmente, a terceira afirmativa define a fase de **projeto de sistema**, na qual cria-se uma representação do domínio do problema do mundo real, levando para um domínio da solução (software), baseando-se nos documentos da Análise. Nesta fase, a preocupação maior é com o “Como”, sem considerar pequenos detalhes de implementação.

**Letra e.**

**035.** (FCC/2013/DPE-SP/AGENTE DE DEFENSORIA – PROGRAMADOR) Com relação aos conceitos básicos e princípios da engenharia de software, analise:

I – Embora nem sempre seja possível uma definição ampla e estável dos requisitos, uma definição de objetivos ambígua pode ser receita para um desastre.

II – Os requisitos de software mudam, mas o impacto da mudança varia dependendo do momento em que ela for introduzida.

III – Se o cronograma de entrega do software atrasar, a solução mais eficiente sempre é a contratação de mais programadores.

IV – Quando diferentes clientes ou usuários propõem necessidades conflitantes é preciso conciliar esses conflitos por meio de um processo de negociação.

Está correto o que se afirma em

- a) I, II, III e IV.
- b) I e IV, apenas.
- c) III e IV, apenas.
- d) II e III, apenas.
- e) I, II e IV, apenas.



Nesta questão, somente a afirmação 3 está incorreta. Nem sempre a contratação de mais programadores é a melhor alternativa, visto que os novos programadores podem demandar algum tempo para entender as regras do sistema em desenvolvimento.

**Letra e.**

**036.** (FCC/2012/TRT – PE) A perspectiva prática sobre o RUP descreve as boas práticas da engenharia de software que são recomendadas para uso no desenvolvimento de sistemas.

Dentre as práticas fundamentais recomendadas incluem-se

- a) utilizar a arquitetura em cascata e efetuar programação em pares.
- b) definir a funcionalidade do protótipo e avaliar o protótipo.
- c) definir o esboço dos requisitos e estabelecer objetivos do protótipo.
- d) utilizar arquiteturas baseadas em componentes e modelar os softwares visualmente.
- e) desenvolver teste inicial a partir de cenários e utilizar frameworks de testes automatizados.



O **Rational Unified Process (RUP)** é um processo de engenharia de software, que oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Busca garantir a produção de software de alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis.

Principais características:

- É **iterativo e incremental**: o software é desenvolvido em várias passadas (iterativo), e cada passada acrescenta uma parte à solução (incremental);
- **Abordagem dirigida por casos de uso**: os casos de uso definidos para o sistema são a fundação para o resto do processo de desenvolvimento;
- **Utiliza arquiteturas baseadas em componentes** (definem uma arquitetura modular; desenvolvidos para reuso; arquiteturas e componentes prontos);
- **Modela o software visualmente** (comunica as decisões sem ambiguidades, ajuda a entender sistemas complexos...);

- É um processo **centrado na arquitetura**: para RUP os aspectos relacionados aos maiores riscos de um projeto de desenvolvimento estão intimamente ligados à arquitetura. Portanto, devemos tratar como centro (core) do nosso desenvolvimento, nossos requisitos arquiteturais do projeto.

**Letra d.**

**037.** (FCC/TJ-SE/2010) De acordo com o RUP, balancear objetivos, administrar riscos e superar restrições para entregar um produto que atenda às necessidades de clientes e usuários é papel do

- a) Gerente de Projetos.
- b) Analista de Sistemas.
- c) Administrador de Dados.
- d) Analista de Requisitos.
- e) Arquiteto de Software.



As atividades mencionadas estão relacionadas ao Gerente de Projetos.

**Letra a.**

**038.** (CESPE/2013/TRT – 10ª REGIÃO (DF E TO)) ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO) As atividades fundamentais relacionadas ao processo de construção de um software incluem a especificação, o desenvolvimento, a validação e a evolução do software.



Um processo de software é um conjunto de atividades e resultados associados que levam à produção de um produto de software.

Nesse contexto, Sommerville (2007, p. 43) destaca as **quatro atividades fundamentais do processo de software**, comuns a todos eles, que são: especificação de software, projeto e implementação de software, validação de software e evolução do software.

#### Especificação de Software

São definidas as funcionalidades do software e restrições para sua operação.

#### Projeto e Implementação de Software

O software que atenda à especificação deve ser produzido.

#### Validação de Software

O software deve ser avaliado para garantir que ele faça o que o cliente deseja.

## Evolução do Software

O software evolui para atender às necessidades de mudança do cliente.

Segundo o autor, essas atividades são organizadas de modo diferente nos diversos processos de desenvolvimento. Como exemplo, no modelo em cascata são organizadas em sequência, ao passo que, no desenvolvimento evolucionário, elas são intercaladas. Como essas atividades serão organizadas dependerá do tipo de software, pessoas e estruturas organizacionais envolvidas.

Com algumas variações de nomes entre os principais autores da área (Pressman e Sommerville), é correto destacar que as **quatro atividades básicas do processo de software** são: especificação, desenvolvimento, validação e evolução.

**Certo.**

**039.** (CESPE/2013/TRT – 10ª REGIÃO (DF E TO)/ANALISTA JUDICIÁRIO – TECNOLOGIA DA INFORMAÇÃO) O ciclo de vida de um software, entre outras características, está relacionado aos estágios de concepção, projeto, criação e implementação.



Sommerville (2007, p. 43) destaca as **quatro atividades fundamentais do processos de software**, comuns a todos eles, que são: especificação de software, projeto e implementação de software, validação de software e evolução do software. Segundo o autor, essas atividades são organizadas de modo diferente nos diversos processos de desenvolvimento.

Pressman (2011) destaca que uma metodologia de processo genérica para Engenharia de Software comprehende **cinco** atividades:

- **Comunicação** – Antes de iniciar o trabalho, é de vital importância comunicar-se e colaborar com o cliente (e outros interessados, como: executivos, usuários finais, engenheiros de software, o pessoal de suporte, etc.). A intenção aqui é compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.
- **Planejamento** – Estabelecimento do plano de projeto de software, que descreve as tarefas técnicas a ser conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a ser produzidos e um cronograma de trabalho.
- **Modelagem** – Criação de modelos representativos do software (para melhor entender as necessidades do software e o projeto que irá atender a essas necessidades).
- **Construção** – Essa atividade combina geração de código (manual ou automatizada) e testes necessários para revelar erros na codificação.
- **Entrega (Implantação)** – Entrega do produto ao cliente, que irá avaliá-lo e fornecer feedback baseado na avaliação.

Essas cinco atividades metodológicas genéricas podem ser utilizadas para o desenvolvimento de programas pequenos e simples, para a criação de grandes aplicações para a Internet e para a engenharia de grandes e complexos sistemas baseados em computador (Pressman, 2011). As atividades metodológicas são complementadas pelas atividades de apoio (umbrella activities), como: controle e acompanhamento do projeto, administração de riscos, garantia da qualidade, gerenciamento da configuração de software, dentre outras.

Observe que os 2 autores trabalham com um modelo genérico de processo de software, e não com o ciclo de vida do software. No entanto, conceitualmente falando, o modelo de processo de software não se diferencia do modelo de ciclo de vida, ambos trazem uma descrição, em alto nível, das atividades envolvidas na construção de um software e suas dependências. Ainda, cabe destacar que as sequências de atividades trazidas pelo Pressman e Sommerville são essencialmente iguais e estão falando do mesmo processo: o desenvolvimento de um software. Assim, pode-se destacar que o ciclo de vida de um software, entre outras características, estará relacionado aos estágios de concepção, projeto, criação e implementação.

**Certo.**

---

**040.** (ESAF/AFC/STN/TI/INFRAESTRUTURA/2008) No RUP (Rational Unified Process), a descrição de critérios de aceitação para o projeto ocorre na fase de

- a) Concepção.
- b) Elaboração.
- c) Construção.
- d) Transição.
- e) Testes.



a) **Certa.** A **fase de concepção** contém os *workflows* necessários para que as partes interessadas (*stakeholders*) concordem com os objetivos, arquitetura e o planejamento do projeto. Se as partes interessadas tiverem bons conhecimentos, então, pouca análise será requerida. Caso contrário, uma análise maior será requerida. Como cita o RUP, o ideal é que sejam feitas iterações. Porém estas devem ser bem definidas quanto a sua quantidade e objetivos. Complementando, é na fase de Concepção que deve ser definido o escopo da versão, especificando o produto a ser gerado. É nessa etapa que se avaliará a viabilidade do projeto, pois neste momento os requisitos operacionais e os critérios de aceitação serão apresentados. Devem ser levantados os riscos que possam comprometer o andamento do projeto, levando em consideração questões de arquitetura. É pertinente também a aplicação de um cronograma preliminar. Dentre os produtos finais desta fase estão o cronograma, o

diagrama de caso de uso inicial, o caso de uso do negócio que, quando aplicado em sistemas comerciais, deve fornecer uma estimativa de retorno do investimento e o protótipo preliminar (MAGALHÃES, 2003).

b) **Errada.** A fase de elaboração será apenas para o projeto do sistema, buscando complementar o levantamento / documentação dos casos de uso, voltado para a arquitetura do sistema, revisa a modelagem do negócio para os projetos e inicia a versão do manual do usuário. Deve-se aceitar: visão geral do produto (incremento + integração) verificando se o mesmo está estável; se o plano do projeto é confiável; se os custos são admissíveis.

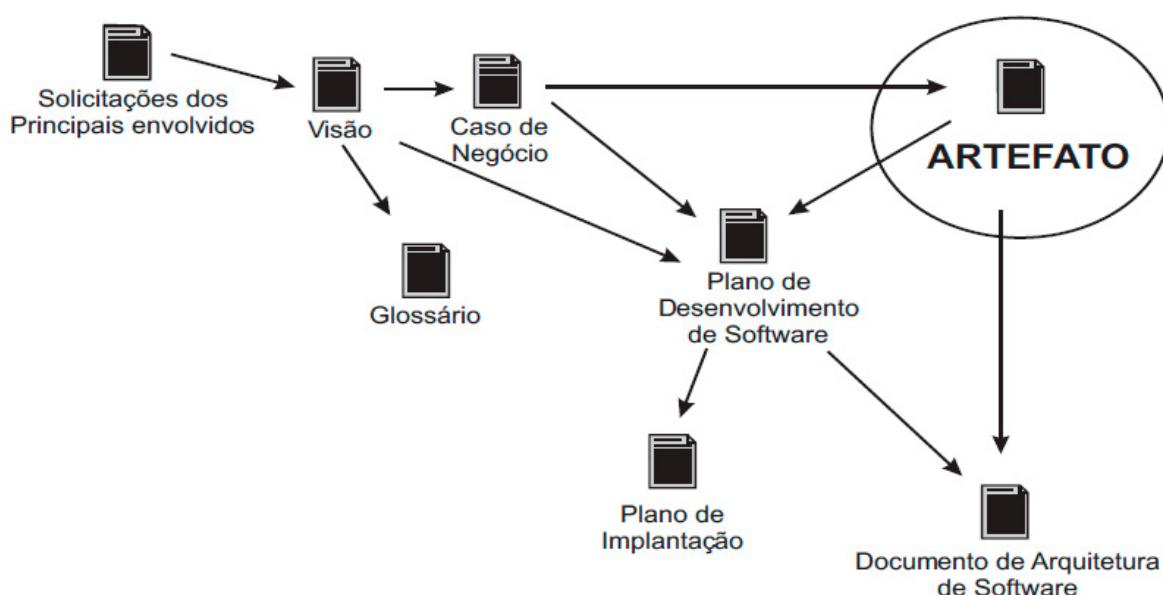
c) **Errada.** Na fase de construção, começa o desenvolvimento físico do software, produção de códigos, testes alfa e beta. Devem-se aceitar testes, e processos de testes estáveis, e se os códigos do sistema constituem “baseline”.

d) **Errada.** Nesta fase ocorre a entrega (“deployment”) do software, é realizado o plano de implantação e entrega, acompanhamento e qualidade do software. Produtos (releases, versões) devem ser entregues, e ocorrer a satisfação do cliente.

e) **Errada.** Não existe uma fase específica de testes no RUP. Os testes são realizados em todas as fases.

**Letra a.**

**041.** (CESGRANRIO/2010/IBGE/ANÁLISE DE SISTEMAS/DESENVOLVIMENTO DE APLICAÇÕES) A figura abaixo apresenta alguns dos principais artefatos do RUP (Rational Unified Process) e o fluxo de informações existentes entre eles.



Qual é o nome do artefato identificado, na figura, pela palavra ARTEFATO e por um círculo?

- a) Projeto do Sistema
- b) Lista de Riscos
- c) Especificação Suplementar
- d) Plano de Teste
- e) Modelo de Casos de Uso



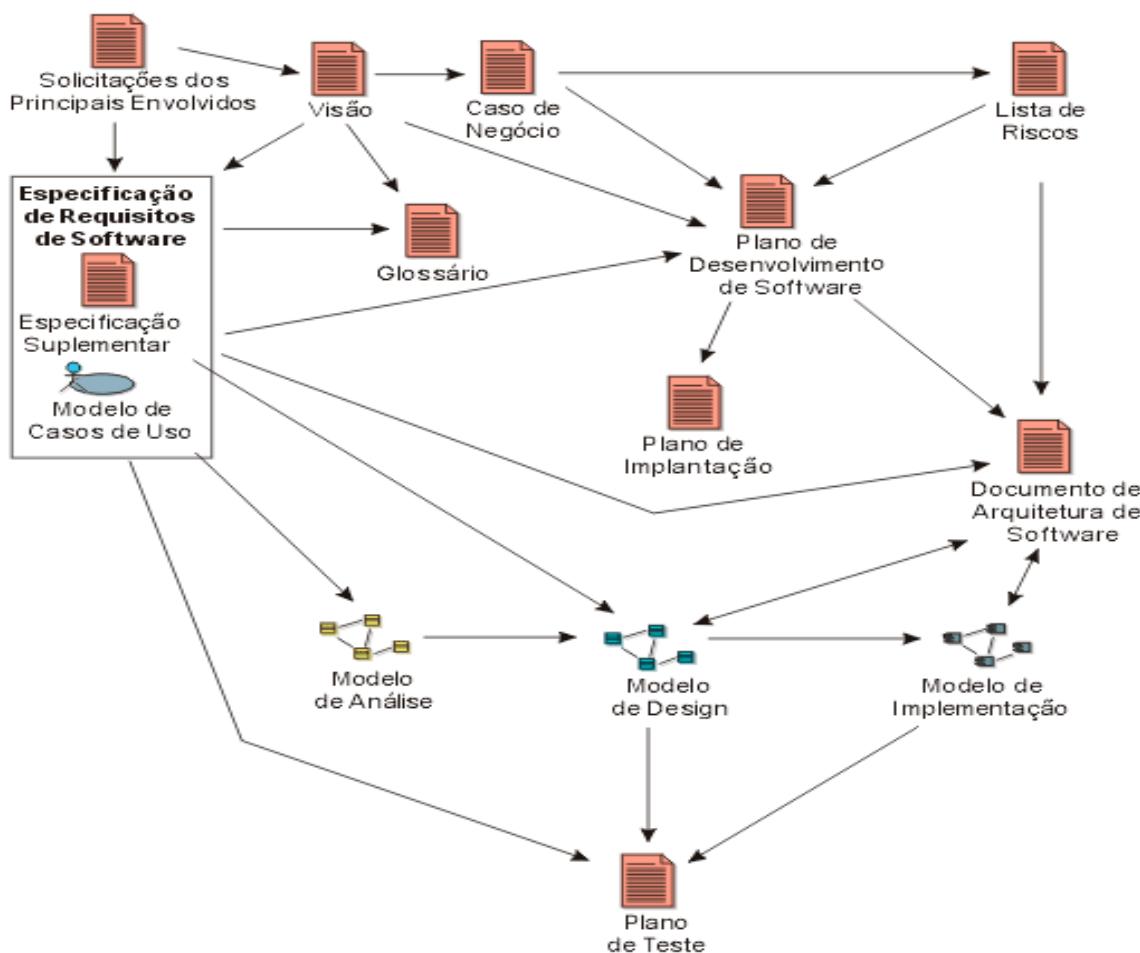
**Artefatos** são produtos de trabalho tangíveis e bem definidos consumidos, produzidos ou modificados por tarefas. Artefatos podem ser compostos por outros artefatos. São exemplos de artefatos: um modelo, como o Modelo de Casos de Uso ou o Modelo de Design.

O **Caso de Negócio** fornece as informações necessárias do ponto de vista de um negócio, para determinar se vale ou não a pena investir no projeto.

O **Plano de Desenvolvimento de Software** é um artefato composto e abrangente que reúne todas as informações necessárias ao gerenciamento do projeto. Ele inclui vários artefatos separados, desenvolvidos durante a Fase de Iniciação, e é mantido durante todo o projeto.

O **Documento de Arquitetura de Software** fornece uma visão geral de arquitetura abrangente do sistema, usando diversas visões de arquitetura para descrever diferentes aspectos do sistema.

Então, podemos concluir que o artefato que fica entre o fluxo de informações dos artefatos indicados é o **Lista de Riscos**, que oferece uma lista de riscos conhecidos e perigosos para o projeto, classificada em ordem decrescente de importância e associada a ações específicas de contingência ou diminuição de riscos (IBM RUP, 2010).



Fonte: (Victorino, 2009)

### Letra b.

**042. (CESGRANRIO/PETROBRAS/ENGENHARIA DE SOFTWARE/2008)** Um princípio fundamental do Processo Unificado é

- ser centrado em arquitetura.
- empregar times auto-dirigidos e auto-organizados.
- o desenvolvimento em cascata.
- a programação em pares.
- a propriedade coletiva do código fonte.

**Obs.:** O Processo Unificado possui características específicas, como ser orientado a casos de uso, ser centrado na arquitetura, ser iterativo e incremental.

Quanto ao princípio “ser centrado em arquitetura” cabe destacar:

- A arquitetura é a visão de todos os modelos que juntos representam o sistema como um todo.

- O conceito de arquitetura de software engloba os aspectos estáticos e dinâmicos mais significantes do sistema. A arquitetura cresce além das necessidades do empreendimento, como percebido pelos usuários e suporte, e refletido nos casos de uso.
- A arquitetura também é influenciada por muitos outros fatores, como a plataforma na qual o software será implantado, os blocos de construção reutilizáveis, requisitos de desenvolvimento e requisitos não funcionais.
- A arquitetura é a visão do projeto do sistema como um todo, destacando suas características mais importantes, mas sem entrar em detalhes.
- O processo ajuda o arquiteto a concentrar-se nas metas corretas, como inteligibilidade, poder de recuperação para mudanças futuras e reutilização.
- A relação existente entre casos de uso e a arquitetura é que os casos de uso estão ligados à funcionalidade de um sistema e a arquitetura, por sua vez está ligada à forma deste.
- Funcionalidade e forma devem estar balanceadas para se alcançar um produto final de qualidade, ou seja, casos de uso e arquitetura devem estar ligados a tal ponto que o primeiro seja desenvolvido de acordo com a arquitetura, e esta por sua vez, forneça um ambiente para a realização de todos os requisitos dos casos de uso.
- A arquitetura de um sistema deve ser projetada a ponto de permitir que o sistema evolua, não apenas durante o início de seu desenvolvimento, mas por meio de gerações futuras.
- Para alcançar este objetivo, os arquitetos devem trabalhar com as funções chaves de um sistema, ou seja, os casos de uso chaves de um sistema.

**Letra a.**

**043.** (ESAF/MPOG/2008) O processo da engenharia de sistemas possui importantes diferenças em relação ao processo de desenvolvimento de software, principalmente no que se refere às suas fases. Indique a opção que representa as fases do processo da engenharia de sistemas.

- a)** Comunicação, Planejamento, Modelagem, Construção e Implantação.
- b)** Análise de requisitos, Projeto do sistema, Projeto do programa, Codificação, Testes de Unidades e de Integração, Teste do sistema, Teste de Aceitação, Operação e Manutenção.
- c)** Definição de Requisitos, Projeto de sistemas e de software, *Implementação e testes de unidades, Integração e testes de sistemas, Operação e manutenção.*
- d)** Análise dos requisitos do sistema, Projeto da arquitetura do sistema, Codificação e testes do sistema, Integração e teste do sistema, Teste de qualificação do sistema, Instalação do sistema e Descontinuação do sistema.
- e)** Definição de requisitos, Projeto do sistema, Desenvolvimento de subsistemas, Integração do sistema, Instalação do sistema, Evolução do sistema e Desativação do sistema.



A **engenharia de sistemas** é uma atividade de especificação, projeto, implementação, validação, implantação e manutenção de sistemas sociotécnicos.

Os engenheiros de sistema são responsáveis pelos softwares, e não somente esses, mas também o hardware, as interações do sistema com os usuários e seu ambiente. Além disso, serviços que o sistema fornece, restrições de operação, criação e utilização, a fim de atingir seu propósito (SOMMERVILLE, 2007).

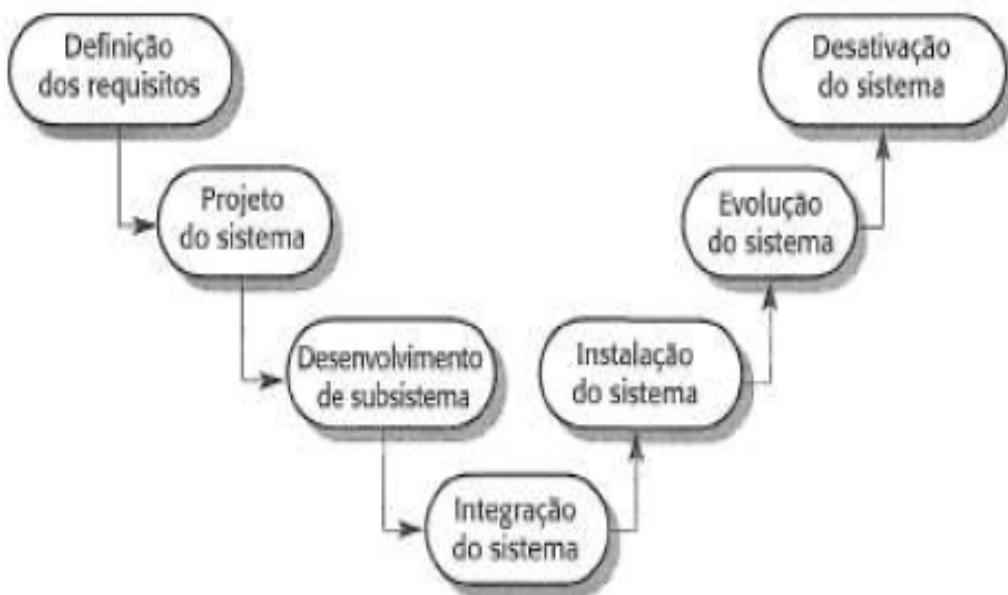


Figura. Fonte: (SOMMERVILLE, 2007)

**Letra e.**

## GABARITO

- 1. E
- 2. E
- 3. E
- 4. c
- 5. C
- 6. b
- 7. E
- 8. b
- 9. b
- 10. d
- 11. e
- 12. b
- 13. b
- 14. b
- 15. c
- 16. a
- 17. b
- 18. C
- 19. C
- 20. e
- 21. b
- 22. C
- 23. c
- 24. b
- 25. d
- 26. c
- 27. e
- 28. c
- 29. C
- 30. E
- 31. a
- 32. c
- 33. C
- 34. e
- 35. e
- 36. d
- 37. a
- 38. C
- 39. C
- 40. a
- 41. b
- 42. a
- 43. e

## **REFERÊNCIAS**

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de Software**. São Paulo: Editora Novatec, 2007.

KRUCHTEN, P. **Introdução ao RUP Rational Unified Process**, 2. ed., Rio de Janeiro: Ciência Moderna, 2003.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**, 7. ed. Porto Alegre: Editora Mc GrawHill, 2011.

PFLEEGER, Shari Lawrence. **Engenharia de Software: Teoria e Prática**. 2.ed., São Paulo: Prentice Hall, 2004.

QUINTÃO, P. L. **Notas de aula da disciplina “Tecnologia da Informação”**. 2021.

QUINTÃO, PATRÍCIA LIMA. **Notas de aula Tecnologia da Informação**. 2021.

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

\_\_\_\_\_. **Engenharia de Software**, 9. ed., São Paulo: Pearson Addison – Wesley, 2011.

CÉSAR, B. **Modelos de desenvolvimento de software: a Catedral e o Bazar**. Disponível em: <<http://www.brod.com.br/node/536>>. Acesso em: 11 jul. 2021.

IBM. Disponível em: <[https://www.ibm.com/developerworks/community/blogs/ctaurion/entry/evolucao\\_do\\_open\\_source?lang=en](https://www.ibm.com/developerworks/community/blogs/ctaurion/entry/evolucao_do_open_source?lang=en)>. Acesso em: 11 jul. 2021.

## Patrícia Quintão



Mestre em Engenharia de Sistemas e computação pela COPPE/UFRJ, Especialista em Gerência de Informática e Bacharel em Informática pela UFV. Atualmente é professora no Gran Cursos Online; Analista Legislativo (Área de Governança de TI), na Assembleia Legislativa de MG; Escritora e Personal & Professional Coach.

Atua como professora de Cursinhos e Faculdades, na área de Tecnologia da Informação, desde 2008. É membro: da Sociedade Brasileira de Coaching, do PMI, da ISACA, da Comissão de Estudo de Técnicas de Segurança (CE-21:027.00) da ABNT, responsável pela elaboração das normas brasileiras sobre gestão da Segurança da Informação.

Autora dos livros: Informática FCC - Questões comentadas e organizadas por assunto, 3<sup>a</sup>. edição e 1001 questões comentadas de informática (Cespe/UnB), 2<sup>a</sup>. edição, pela Editora Gen/Método.

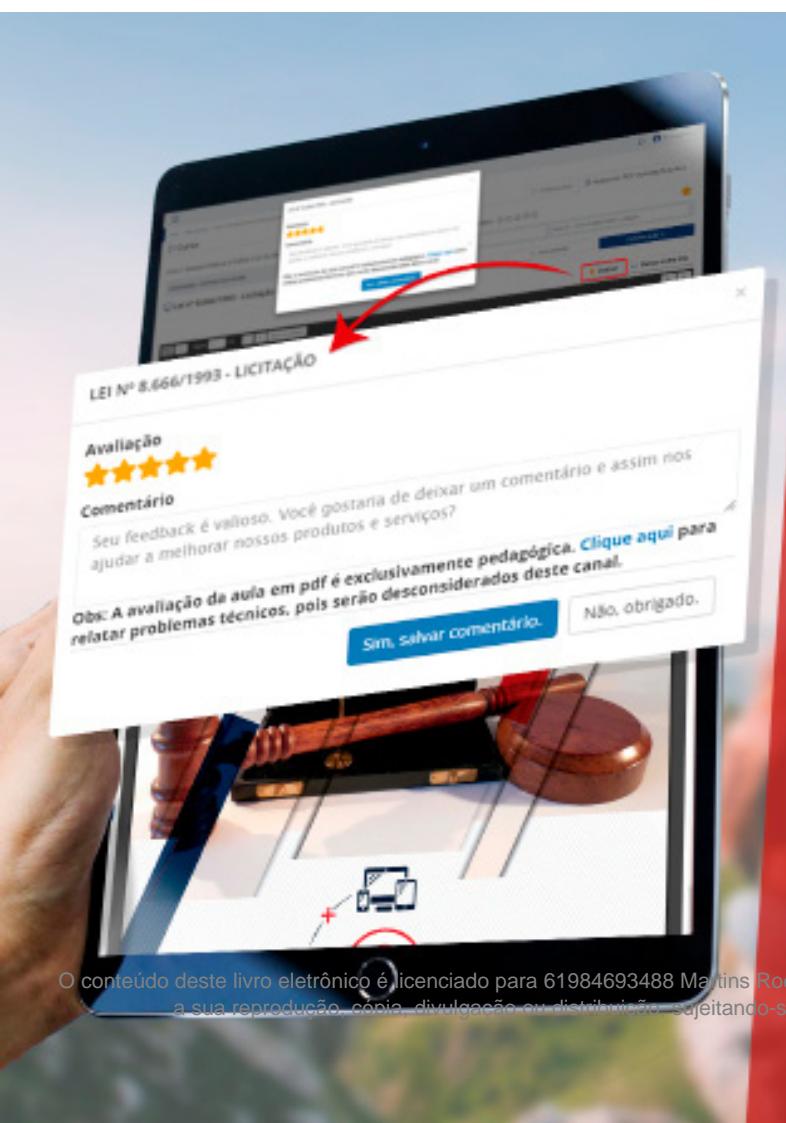
Foi aprovada nos seguintes concursos: Analista Legislativo, na especialidade de Administração de Rede, na Assembleia Legislativa do Estado de MG; Professora titular do Departamento de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia; Professora substituta do DCC da UFJF; Analista de TI/Suporte, PRODABEL; Analista do Ministério Público MG; Analista de Sistemas, DATAPREV, Segurança da Informação; Analista de Sistemas, INFRAERO; Analista - TIC, PRODEMGE; Analista de Sistemas, Prefeitura de Juiz de Fora; Analista de Sistemas, SERPRO; Analista Judiciário (Informática), TRF 2<sup>a</sup> Região RJ/ES, etc.

@coachpatriciaquintao

/profapatriciaquintao

@plquintao

t.me/coachpatriciaquintao



# NÃO SE ESQUEÇA DE AVALIAR ESTA AULA!

SUA OPINIÃO É MUITO IMPORTANTE  
PARA MELHORARMOS AINDA MAIS  
NOSSOS MATERIAIS.

ESPERAMOS QUE TENHA GOSTADO  
DESTA AULA!

PARA AVALIAR, BASTA CLICAR EM LER  
A AULA E, DEPOIS, EM AVALIAR AULA.

**AVALIAR**