Estratégia Concursos

# Aula 00

*Bizu Estratégico p/ ALCE (Analista Legislativo - Informática) - Pós-Edital*

Autor:

17 de Junho de 2020

# BIZU ESTRATÉGICO – ENGENHARIA DE SOFTWARE – ALCE

Fala, pessoal. Tudo certo?

Neste material, trazemos uma seleção de bizus da disciplina de **Engenharia de Software** para o concurso de **Analista Legislativo (Informática)** da **Assembléia Legislativa do Ceará**.

O objetivo é proporcionar uma revisão rápida e de alta qualidade aos alunos por meio de tópicos desse conteúdo com base no grau de incidência em provas.

Espero que gostem do que preparamos para vocês, pois o conteúdo foi selecionado com o objetivo de vocês acertarem o maior número de questões.

**Coach Luciano Duarte**

@proflucianoduarte

**Coach Leonardo Mathias**

@profleomathias

| Engenharia de Software | | |
|---|---|---|
| Assunto | Bizus | Caderno de Questões |
| Modelos de Ciclo de Vida | 1 a 9 | https://bit.ly/2XGTdQU |
| Metodologias Ágeis | 10 a 12 | https://bit.ly/2Yc7bsV |
| Engenharia de Requisitos | 13 e 14 | https://bit.ly/2XFHPod |
| Análise e Projeto Orientado a Objetos | 15 e 16 | https://bit.ly/2Mziq9E |
| Qualidade de Software e Testes | 17 a 24 | https://bit.ly/377fScg |
| Arquitetura de Software | 25 e 26 | https://bit.ly/2UkaeOy |
| SOA, Web Services e REST | 27 a 29 | https://bit.ly/30i9gpT |
| Engenharia de Usabilidade | 30 | https://bit.ly/3eZy8H7 |
| Lei 13.709/2018 | 31 e 32 | https://bit.ly/3eYkLag |

# ANÁLISE ESTATÍSTICA

Vejamos uma análise estatística para sabermos quais são os assuntos mais exigidos pela banca CEBRASPE no âmbito da disciplina de Engenharia de Software:

| Engenharia de Software (foram encontradas 505 questões) | | |
|---|---|---|
| Assunto | Quantidade de Questões | % |
| Modelos de Ciclo de Vida | 25 | 5% |
| Metodologias Ágeis | 101 | 20% |
| Engenharia de Requisitos | 81 | 16% |
| Análise e Projeto Orientado a Objetos | 78 | 15% |
| Qualidade de Software e Testes | 69 | 14% |
| Arquitetura de Software | 37 | 7% |
| SOA, Web Services e REST | 88 | 17% |
| Engenharia de Usabilidade | 24 | 5% |
| Lei 13.709/2018 | 2 | 0% |

# Modelos de Ciclo de Vida

## 1. Modelo de ciclo de vida

i. Trata-se das fases pelas quais um software passa desde o seu início até o seu fim e como essas fases se relacionam.

## 2. Fases mais comuns dos modelos de ciclo de vida

i. Planejamento
   - ➢ Possui o objetivo de fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos.

ii. Análise e Especificação de Requisitos
   - ➢ O escopo deve ser refinado. Os requisitos devem ser modelados, avaliados e documentados.

iii. Projeto
   - ➢ Envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado.

iv. Implementação
   - ➢ O projeto deve ser traduzido para uma forma passível de execução pela máquina.

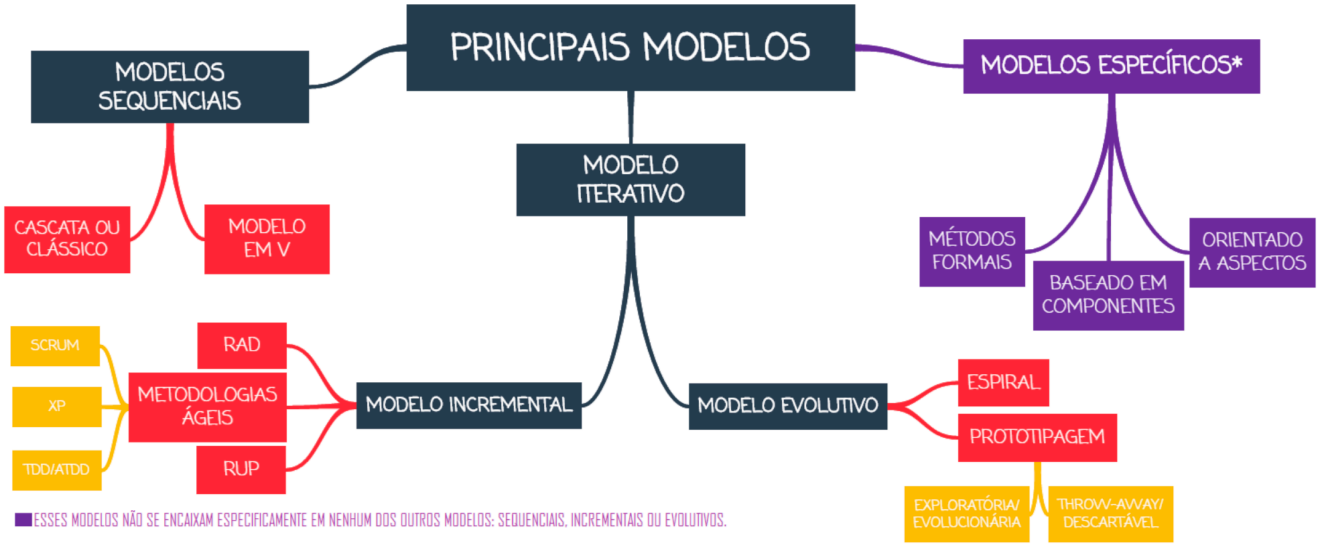
v. Testes

vi. Entrega e Implantação
   - ➢ O propósito desta fase é estabelecer que o software satisfaz os requisitos dos usuários.

vii. Operação
   - ➢ O software é utilizado pelos usuários no ambiente de produção.

viii. Manutenção

## 3. Principais Modelos




## 4. Modelo em Cascata

**i.** Outros nomes

- Clássico, Sequencial, Linear, Tradicional e Waterfall.

**ii.** Principais características

- Uma fase só se inicia após o término e aprovação da fase anterior.
- Sua utilização deve ocorrer preferencialmente quando os requisitos forem bem compreendidos e houver pouca probabilidade de mudanças radicais durante o desenvolvimento do sistema.

**iii.** Vantagens e desvantagens

| VANTAGENS | DESVANTAGENS |
|---|---|
| É simples de entender e fácil de aplicar, facilitando o planejamento. | Divisão inflexível do projeto em estágios distintos. |
| Fixa pontos específicos para a entrega de artefatos. | Dificuldade em incorporar mudanças de requisitos. |
| Funciona bem para equipes tecnicamente fracas. | Clientes só visualizam resultados próximos ao final do projeto. |
| É fácil de gerenciar, devido a sua rigidez. | Atrasa a redução de riscos. |
| Realiza documentação extensa por cada fase ou estágio. | Apenas a fase final produz um artefato de software entregável. |
| Possibilita boa aderência a outros modelos de processo. | Cliente deve saber todos os requisitos no início do projeto. |
| Funciona bem com projetos pequenos e com requisitos bem conhecidos. | Modelo inicial (Royce) não permitia feedback entre as fases do projeto. |
| | Pressupõe que os requisitos ficarão estáveis ao longo do tempo. |
| | Não funciona bem com projetos complexos e OO, apesar de compatível. |

## 5. Modelo Iterativo e Incremental

i. Principais características
   - ➢ O cliente pode receber e avaliar as entregas do produto mais cedo, já no início do desenvolvimento do software.
   - ➢ Acomoda melhor as mudanças – aumentando o reuso e a qualidade.

## 6. RAD - Rapid Application Development.

i. Modelo iterativo e incremental que enfatiza o ciclo de desenvolvimento curto (60 a 90 dias).
ii. Utiliza o reúso de componentes à exaustão.
iii. Há uma interação direta e intensa com o usuário.

## 7. Modelo Iterativo e Evolucionário

i. Desenvolvimento de uma implementação inicial, expondo o resultado aos comentários do usuário e refinando esse resultado por meio de várias versões.

## 8. Prototipagem

i. É utilizada quando não se conhece bem os requisitos.
ii. Processo iterativo, interativo e rápido de desenvolvimento.
iii. O protótipo serve como um mecanismo de identificação dos requisitos do software.
iv. Desvantagem: usuários confundem protótipo com o sistema final.

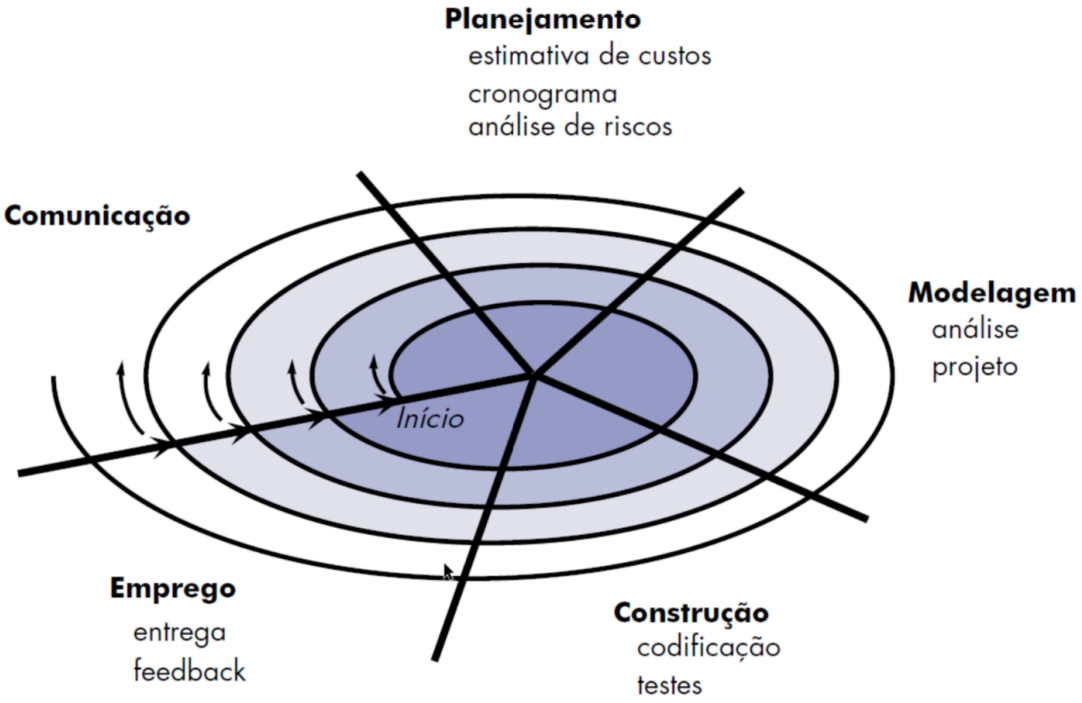


## 9. Modelo em espiral

i. Combina atividades de desenvolvimento com aspectos gerenciais.

ii. Ao final de cada loop na espiral, deve-se decidir se o projeto continuará ou se será interrompido.

iii. Enfatiza bastante a Análise de Riscos.



# Metodologias Ágeis

## 10. Conceitos Básicos

| | | |
|---|---|---|
| INDIVÍDUOS E INTERAÇÃO | > | PROCESSOS E FERRAMENTAS |
| SOFTWARE EM FUNCIONAMENTO | > | DOCUMENTAÇÃO ABRANGENTE |
| COLABORAÇÃO COM CLIENTE | > | NEGOCIAÇÃO DE CONTRATOS |
| RESPONDER A MUDANÇAS | > | SEGUIR UM PLANO |

i. Mesmo havendo valor nos itens à direita, valorizam-se mais os itens à esquerda.

## 11. Scrum

i. **Sprint**

- Dura um mês ou menos.
- Durante a Sprint, é proibido realizar mudanças que coloquem em risco os objetivos da própria Sprint.
- É proibido mudar a composição da equipe ou diminuir as metas de qualidade.

ii. **Papeis**

- Product Owner
  - Único que pode gerenciar o Product Backlog.
  - Responsável por priorizar/ordenar os itens do Product Backlog.
  - Único que pode cancelar uma Sprint.
- Scrum Master
  - Responsável por garantir que o Scrum seja entendido e aplicado.
  - Responsável pela gestão de pessoas e gestão do processo.
  - Responsável por ajudar a remover impedimentos que o time enfrente.
- Development Team
  - São multifuncionais.
  - Responsável pela criação do produto.

iii. **Artefatos**

- Product Backlog
  - Lista ordenada de requisitos ou funcionalidades que o produto deve conter.

- Somente o Product Owner pode inserir, reordenar ou remover itens.
- A lista é dinâmica e é a origem única dos requisitos.

➢ Sprint BackLog
- Conjunto de itens que serão implementados na próxima Sprint.
- Somente os membros do time de desenvolvimento podem adicionar novos itens.

➢ Product Increment
- Ao final de cada sprint, um Product Increment é entregue.

**iv. Eventos**

➢ Sprint Planning
- Evento para planejar o trabalho a ser realizado na Sprint.

➢ Daily Meeting
- Reunião para planejar o trabalho das próximas 24 horas e inspecionar o trabalho desde a última reunião.

➢ Sprint Review
- Possui o objetivo de inspecionar o que a equipe de desenvolvimento produziu e adaptar o plano para a Sprint seguinte.
- Foco no produto.

➢ Sprint Retrospective
- Foco nas melhorias do processo.

## 12. Extreme Programming (XP)

**i.** Principais práticas

➢ Planejamento incremental
➢ Pequenos releases
➢ Desenvolvimento Test-First
- Primeiro escreve o teste, depois faz-se a implementação.

➢ Refactoring
➢ Programação em pares
➢ Propriedade coletiva
➢ Integração contínua
➢ Metáforas
- A equipe se comunica por meio de metáforas para facilitar a comunicação.

➢ Client On-Site

- Um representante do usuário final deve estar disponível em tempo integral.

- Reuniões em pé
- Jogo do planejamento
  - O planejamento é feito com base nas histórias dos usuários. Os clientes priorizam e os desenvolvedores avaliam e estimam.

# Engenharia de Requisitos

## 13. Classificações

**i. Quanto ao nível de abstração**

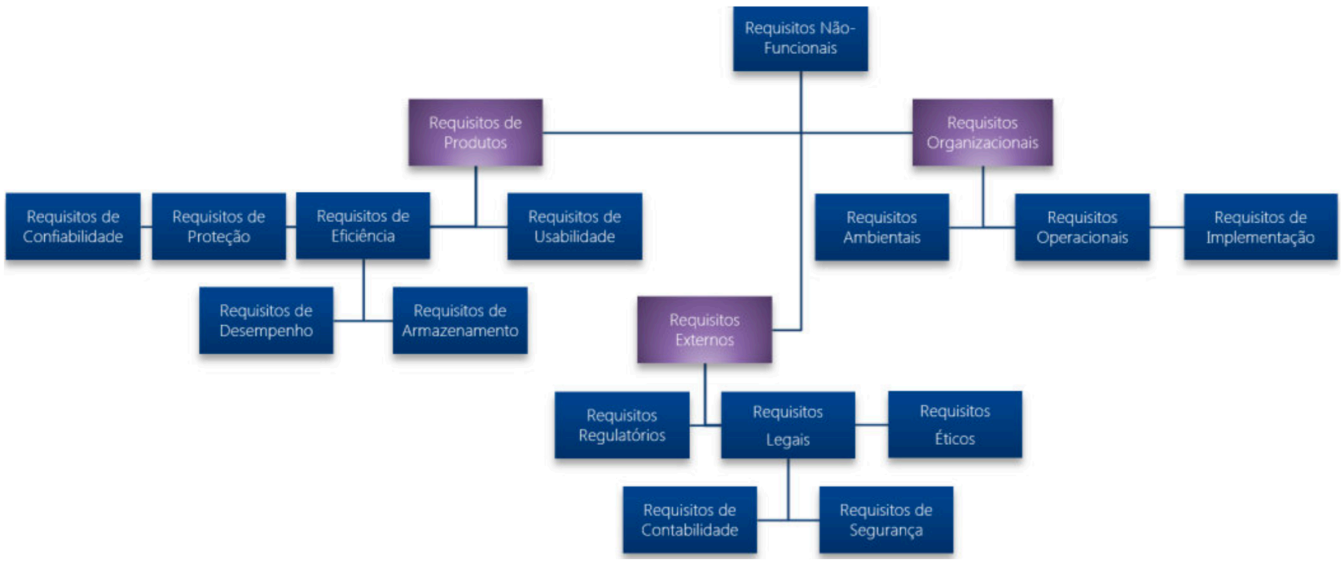
- Requisitos de Usuário: alto nível de abstração e poucos detalhes.
- Requisitos de Sistema: baixo nível de abstração e muitos detalhes.

**ii. Quanto à evolução**

- Requisitos Permanentes: mudam pouco ou demoram para mudar.
- Requisitos Voláteis: se modificam quando o sistema está em desenvolvimento ou em uso.

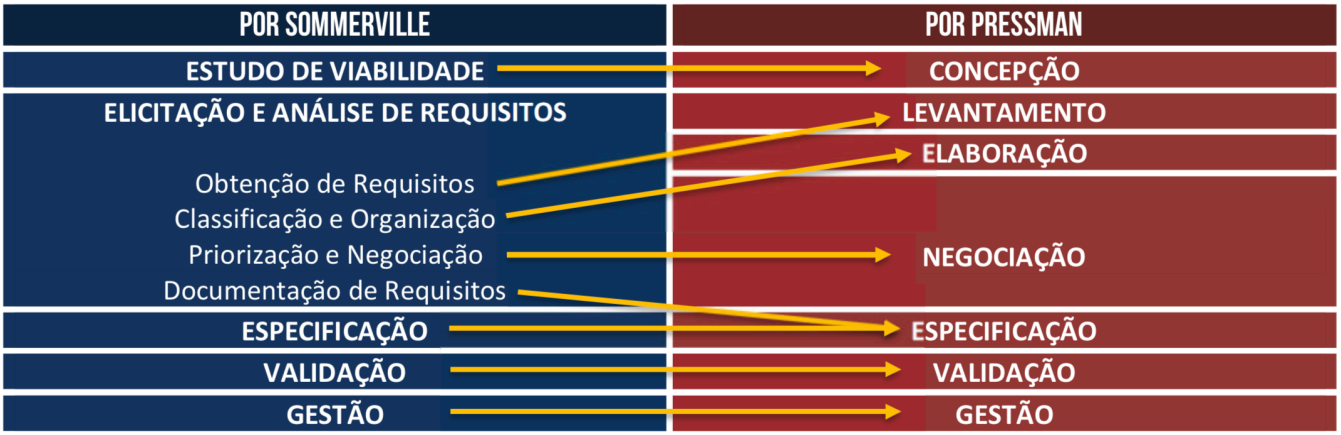
**iii. Quanto à funcionalidade**

- Requisitos Funcionais: são ações ou funcionalidades que o sistema deve fornecer para atingir seus objetivos.
- Requisitos Não Funcionais: são restrições ou condições estipuladas sobre as quais o sistema deve funcionar.



## 14. Processo de Engenharia de Requisitos

### i. Fases










### ii. Técnicas

- Entrevistas
- Etnografia
  - Técnica de observação. O analista fica dentro do campo de atuação dos usuários, observando o trabalho diário.
- Cenários
- Questionários
  - Formulários com questões pré-definidas.
- Workshop
  - Reunião estruturada.
- Brainstorming
- Prototipação
- Histórias de Usuários
  - História contada na linguagem do usuário final.

# Análise e Projeto Orientado a Objetos

## 15. Paradigma Orientado a Objetos

**i. Classes**

- Modelo a partir do qual os objetos são construídos.

**ii. Objetos**

- São instâncias das classes
- Componentes: identidade, estado (propriedade) e comportamento (operações).

**iii. Atributos**

- Atributo de Classe: variável cujo valor é comum a todos os objetos de uma classe.
- Atributo de Instância: variável cujo valor é específico ao objeto.

**iv. Métodos**

- Descrições das operações que um objeto executa quando recebe uma mensagem.
- Construtor: método especial que são chamados automaticamente quando instâncias são criadas.
- Método de Classe: método que não é relativo a nenhuma instância particular.
- Método de Instância: método que realiza operações específicas para um objeto.
- Método abstrato: método para o qual não é definida uma forma de implementação.

**v. Abstração**

- Classe Abstrata
  - Não pode ser instanciada diretamente.
  - É compartilhada por um grupo de classes derivadas.
  - Pode ter métodos concretos.
- Classe Concreta
  - Pode ser instanciada diretamente.
  - Não pode conter métodos abstratos.
- Interface
  - Todos os métodos devem ser abstratos.

**vi. Encapsulamento**

- É uma forma de restringir o acesso ao comportamento interno de um objeto.
- Especificadores ou modificadores de acesso: definem a visibilidade de um atributo, método ou classe.

| MODIFICADOR/ESPECIFICADOR | | | CLASSE | PACOTE | SUBCLASSE | TODOS |
|---|---|---|---|---|---|---|
| UML | PÚBLICO | + | X | X | X | X |
| | PROTEGIDO | # | X | | X | |
| | PACOTE | ~ | X | X | | |
| | PRIVADO | – | X | | | |

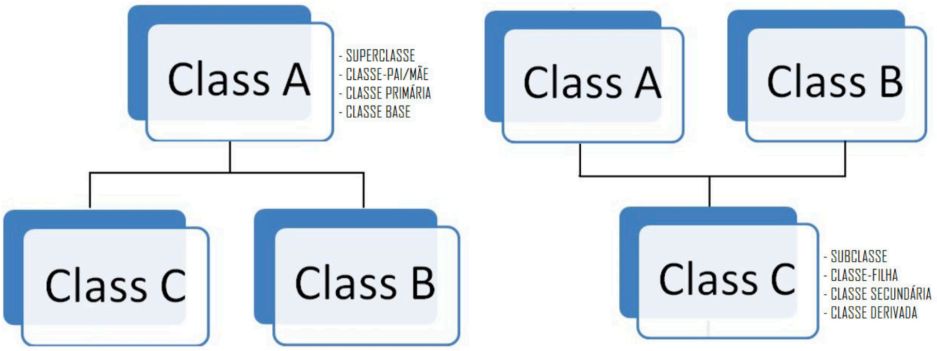
| MODIFICADOR/ESPECIFICADOR | | | CLASSE | PACOTE | SUBCLASSE | TODOS |
|---|---|---|---|---|---|---|
| JAVA | PÚBLICO | + | X | X | X | X |
| | PROTEGIDO | # | X | X | X | |
| | DEFAULT | ~ | X | X | | |
| | PRIVADO | - | X | | | |

**vii. Polimorfismo**

- Permite que a mesma mensagem seja enviada a diferentes objetos e cada objeto execute a operação que é mais apropriada para a sua classe.

**viii. Herança**

- Mecanismo que permite que classes compartilhes atributos e métodos.
- Herança múltipla: Quando uma subclasse herda diretamente de duas ou mais superclasses.




## 16. Análise e Projeto

**i. Análise**

- Modelagem do problema
- Foco no cliente

**ii. Projeto**

- Modelagem da solução
- Foco no programador

# Qualidade de Software e Testes

## 17. Fatores de Qualidade

| FATORES | DESCRIÇÃO |
|---|---|
| CORREÇÃO | O quanto um programa satisfaz a sua especificação e atende aos objetivos da missão do cliente. |
| CONFIABILIDADE | O quanto se pode esperar que um programa realize a função pretendida com a precisão exigida. |
| EFICIÊNCIA | A quantidade de recursos computacionais e código exigidos por um programa para desempenhar sua função. |
| INTEGRIDADE | O quanto o acesso ao software ou dados por pessoas não autorizadas pode ser controlado. |
| USABILIDADE | Esforço necessário para aprender, operar, preparar a entrada de dados e interpretar a saída de um programa. |
| FACILIDADE DE MANUTENÇÃO | Esforço necessário para localizar e corrigir um erro em um programa. |
| FLEXIBILIDADE | Esforço necessário para modificar um programa em operação. |
| TESTABILIDADE | Esforço necessário para testar um programa de modo a garantir que ele desempenhe a função destinada. |
| PORTABILIDADE | Esforço necessário para transferir o programa de um ambiente de hardware e/ou software para outro. |
| REUSABILIDADE | O quanto um programa [ou partes de um programa] pode ser reutilizado em outras aplicações. |
| INTEROPERABILIDADE | Esforço necessário para integrar um sistema a outro. |

## 18. Garantia e Controle de Qualidade

| GARANTIA DE QUALIDADE | CONTROLE DE QUALIDADE |
|---|---|
| Garantia da qualidade garante que o processo é definido e apropriado. | As atividades de controle da qualidade focam na descoberta de defeitos específicos. |
| Metodologia e padrões de desenvolvimento são exemplos de garantia da qualidade. | Um exemplo de controle da qualidade poderia ser: *"Os requisitos definidos são os requisitos certos?"* |
| Garantia da qualidade é orientada a processo. | Controle da qualidade é orientado a produto. |
| Garantia da qualidade é orientada a prevenção. | Controle da qualidade é orientado a detecção. |
| Foco em monitoração e melhoria de processo. | Inspeções e garantia de que o produto de trabalho atenda aos requisitos especificados. |
| As atividades são focadas no início das fases no ciclo de vida de desenvolvimento de software. | As atividades são focadas no final das fases no ciclo de vida de desenvolvimento de software. |
| Garantia da qualidade garante que você está fazendo certo as coisas e da maneira correta. | Controle da qualidade garante que os resultados do seu trabalho são os esperados conforme requisitos. |

## 19. Verificação e Validação

### i. Verificação

- Depende da especificação
- Estamos construindo o produto corretamente?
- O produto está de acordo com as especificações do sistema?
- Verificação estática
  - Inspeção de Software
  - Ocorre sem a necessidade de executar o software
- Verificação dinâmica
  - Teste
  - Envolve executar o software

### ii. Validação

- Depende do usuário
- Estamos construindo o produto correto?
- O produto está de acordo com as necessidades do usuário?

## 20. Princípios fundamentais

i. Testes demonstram a presença de defeitos
ii. Testes exaustivos são impossíveis
iii. Teste o mais breve possível
iv. Agrupem os defeitos mais sensíveis
v. Paradoxo do pesticida
vi. Testes dependem do contexto
vii. Ausencia de defeitos é uma ilusão

## 21. Dimensões dos Testes



## 22. Estratégias / Níveis de Testes

i. **Testes de Unidade**
   - Testa a menor unidade de projeto do software: componente ou módulo.
   - Foca na logica interna dentro dos limites de um componente.

ii. **Testes de Integração**
   - O foco está no projeto e construção da arquitetura de software.

iii. **Testes de Validação (ou Aceitação)**
   - Proporciona a garantia final de que o software satisfaz a todos os requisitos informativos, funcionais, comportamentais e de desempenho.

iv. **Testes de Sistema**
   - O software e outros elementos são testados como um todo.
   - A finalidade é exercitar totalmente o sistema.

## 23. Técnicas de Teste

i. **Teste Caixa-Branca**
   - É necessário ter conhecimento sobre o funcionamento interno dos componentes.
   - Principais técnicas: Testes de Caminho Básico e Testes de Estruturas de Controle.

ii. **Teste Caixa-Preta**
   - Deriva casos de teste a partir da especificação de requisitos, ignorando detalhes de implementação.
   - Principais técnicas: Testes Baseadps em Grafos, Particionamento de Equivalências, Análise de Valor Limite e Teste de Matriz Ortogonal.

iii. **Teste Caixa-Cinza**
   - Versão híbrida entre os Testes Caixa-Branca e Testes Caixa-Preta.

## 24. Tipos de Testes

i. **Teste de Desempenho**
   - Testa o desempenho em tempo de execução do software dentro do contexto de um sistema integrado.
   - Trata do esforço em assegurar que o sistema computacional pode operar sob a carga de operação especificada.

ii. **Teste de Estresse**
   - Envolve o teste para além da capacidade operacional normal.

iii. **Teste de Carga**
   - Procuram determinar como o software responderá a várias condições de carga.

iv. **Teste de Regressão**

- É a reexecução do mesmo subconjunto de testes que já foram executados para assegurar que as alterações não tenham propagado efeitos colaterais indesejados.

v. **Testes Alfa**

- Testes de aceitação.
- Ocorrem no ambiente do desenvolvedor.
- É executado por clientes e usuários finais.

vi. **Testes Beta**

- Testes de aceitação.
- Ocorrem nas instalações dos usuários.
- Executado por clientes e usuários finais.

# Arquitetura de Software

## 25. Coesão e Acoplamento

i. Uma boa arquitetura de software deve ter componentes de projeto com baixo acoplamento e alta coesão.

ii. **Coesão**

- Divisão de responsabilidades

iii. **Acoplamento**

- Dependência de componentes

## 26. Arquitetura em camadas

i. **Arquitetura Cliente-Servidor**

- Duas camadas: Cliente e Servidor

ii. **Arquitetura de três camadas**

- Camada de Apresentação
  - Objetivo de exibir informações ao usuário e traduzir ações do usuário em requisições às demais partes dos sistemas.
- Camada de Negócio
  - Implementam a regra de negócio.
- Camada de Dados
  - Classes que se comunicam com outros sistemas para realizar tarefas ou adquirir informações para o sistema.

iii. **Arquitetura MVC**

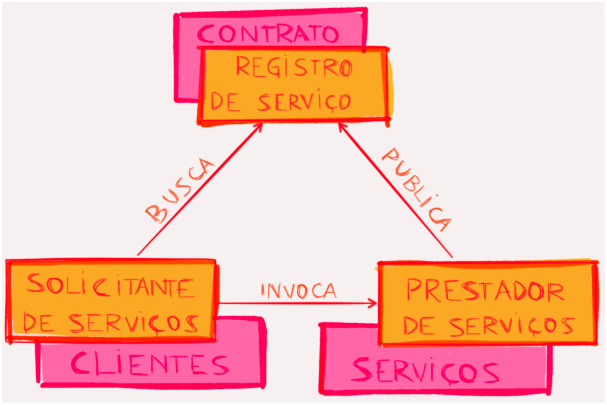
- Modelo
  - Responsável pela representação dos dados.
  - Encapsula as principais funcionalidades e dados do sistema.
- Controle
  - Capaz de enviar comandos para o modelo atualizar o seu estado.
  - Atende às requisições do usuário e seleciona o modelo e a visão que o usuário usará para interagir com o modelo.
- Visão
  - Responsável pela interação com o usuário, sendo responsável apenas pela exibição de dados.

# SOA, Web Services e REST

## 27. SOA

### i. Modelo Triangular

- Prestadores de serviços publicam serviços no registro de serviço; consumidores de serviços realizam uma busca por algum serviço de interesse. Caso o consumidor encontre o serviço desejado, é criado um contrato e devolve-se um endereço (comumente chamado de endpoint) para utilização do serviço.




### ii. Princípios de design

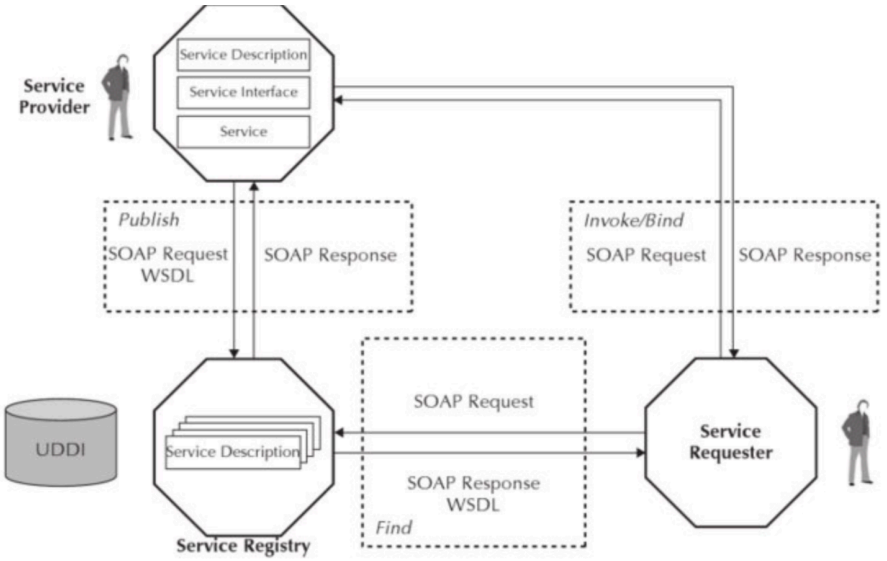
- Contrato de Serviço padronizado
- Baixo Acoplamento de Serviços
- Abstração de Serviços
  - Serviços devem ser vistos como uma caixa preta
- Reusabilidade de Serviços
- Autonomia de Serviços
- Independência de Estados de Serviços
- Visibilidade de Serviços
- Composição de Serviços

### iii. Composição de Serviços

- Orquestração
  - Um processo de negócio centralizado é responsável por controlar e coordenar as interações entre diferentes serviços.
- Coreografia
  - Não há um coordenador central. Cada serviço sabe exatamente quando executar suas operações e com quem interagir.

## 28. Web Services

i. Web Services são componentes de aplicativos baseados em XML, autocontidos e autodescritivos, que se comunicam usando protocolos abertos.




ii. **SOAP**

- Protocolo, escrito em XML, projetado para invocar aplicações remotas em um ambiente independente de plataforma.

iii. **WSDL**

- Linguagem, baseada em XML, utilizada para descrever um Web Service.

iv. **UDDI**

- Especificação técnica para descrever, descobrir e integrar Web Services.

## 29. REST

i. É um estilo arquitetural para construir sistemas fracamente acoplados.

ii. Stateless

| SOAP [SIMPLE OBJECT ACCESS PROTOCOL] | REST [REPRESENTATIONAL STATE TRANSFER] |
|---|---|
| - É um protocolo de comunicação. | - É um estilo arquitetural ou uma técnica de engenharia de software. |
| - Utiliza um Envelope (Cabeçalho + Corpo) enviado por HTTP (FTP, SMT, etc) para transmitir dados. | - Utiliza os recursos oferecidos nativamente pelo HTTP (apesar de poder utilizar outros protocolos). |
| - Suporta somente o formato XML. | - Suporta XML, JSON, YAML, TXT, etc |
| - Apresenta desempenho e escalabilidade menor, devido ao alto overhead. | - Apresenta desempenho e escalabilidade maior, devido ao baixo overhead. |
| - Não permite fazer Caching. | - Permite fazer Caching. |
| - JavaScript pode chamar SOAP, mas é difícil de implementar. | - JavaScript pode facilmente chamar REST. |

# Engenharia de Usabilidade

## 30. Conceitos

**i.** Componentes da qualidade que fazem algo ser usável

| CRITÉRIO | DESCRIÇÃO |
|---|---|
| LEARNABILITY (FACILIDADE DE APRENDIZADO) | Também conhecida como Intuitividade, o sistema deve ser de fácil aprendizado e assimilação, de forma que um usuário – mesmo sem experiência – possa rapidamente obter algum resultado satisfatório. Trata-se do atributo mais importante de acordo com Jakob Nielsen. |
| EFFICIENCY (EFICIÊNCIA) | O sistema deve ser eficiente no uso, isto é, uma vez que o usuário tenha aprendido o design do sistema, ele deve poder executar tarefas com um alto nível de produtividade. |
| MEMORABILITY (MEMORIZAÇÃO) | O sistema deve ser fácil de lembrar, para que o usuário seja capaz de retornar depois de um tempo sem ter que aprender tudo novamente. |
| ERRORS (ERROS) | O sistema deve ter um baixo índice de erros, e se existirem devem permitir uma recuperação rápida e fácil. Ademais, erros catastróficos não devem ocorrer. |
| SATISFACTION (SATISFAÇÃO) | O sistema deve ser agradável de usar, para que os usuários fiquem satisfeitos e voltem mais vezes para usá-lo. |

**ii. Qualidades de interface de usuário**

- Usabilidade
  - Facilidade de aprender a utilizar o sistema, facilidade de lembrar a forma de utilização, eficiência, produtividade, flexibilidade.
- Navegabilidade
  - Propriedade que a interface possui de facilitar ao usuário chegar ao seu destino da maneira mais eficiente possível.
- Comunicabilidade
  - Capacidade de os usuários entenderem o design tal como concebido pelos projetistas.

# Lei 13.709/2018

## 31. Conceitos

i. Lei Geral de Proteção de Dados Pessoais.

ii. Traz regras para disciplinar a forma como os dados pessoais dos indivíduos podem ser armazenados por empresas ou mesmo por outras pessoas físicas.

iii. Dado pessoal

- Informação relacionada a pessoa natural identificada ou identificável.

iv. Dado pessoal sensível

- Dado pessoal sobre origem racial ou étnica, convicção religiosa, opinião política, filiação a sindicato ou organização de caráter religioso, filosófico ou político, dado referente à saúde ou à vida sexual, dado genético ou biométrico quando vinculado a uma pessoa natural.

v. Titular

- Pessoa natural a quem se referem os dados pessoais que são objetos de tratamento.

vi. Controlador

- Pessoa natural ou jurídica, de direito público ou privado, a quem compete as decisões referentes aoo tratamento de dados pessoais.

## 32. Princípios

i. Finalidade

ii. Adequação

iii. Necessidade

iv. Livrre Acesso

v. Qualidade dos Dados

vi. Transparência

vii. Segurança

viii. Prevenção

ix. Não Discriminação

x. Responsabilização e Prestação de Contas

# ESSA LEI TODO MUNDO CONHECE:
# PIRATARIA É CRIME.

**Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.**

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.



**Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.**