REVISÃO PSS

\*REUSO DE SOFTWARE

- Prática que sempre deve ser adotada, visto que influencia diretamente na qualidade, produtividade e reduz custos ao desenvolver o software. Outro ganho é no que diz respeito à manutenção, vale ressaltar que cada profissional/programador que mexe em um determinado algorítmo vai cada vez mais aprimorando-o, corrigindo bugs e tonando-o cada vez mais estável e confiável.

- Não deve-se haver dúvidas quanto ao REUSO, ele é importantíssimo... Os líderes/gerentes/gestores devem incentivar esta prática dentro da empresa, mesmo que isso demande mais tempo. Idéias, códigos fontes e projetos podemf acilmente ser reusados e dar mais performance num projeto futuro.

- Uma dificuldade é no que diz respeito à mentalidade das pessoas envolvidas com o processo de reuso, porém, essas pessoas devem ser treinadas a desenvolver com reuso e para o reuso dos demais. Em determinadas empresas não há incentivo com relação a essa prática tão importante.

\*\*TIPOS DE REUSO\*\*

\*REUSO VERTICAL

- É quando ocorre dentro de um mesmo dómínio, estudase desenvolver um modelo genérico que poderá ser utilizado futuramente.

\*REUSO HORIZONTAL

- Utilização do código em diferentes domínios. Ex: biliotecas matemáticas, strings, etc...

\*REUSO PLANEJADO

- Diretrizes e procedimentos são levantatos, definidos e devem ser seguidos. Requer mais investimento e comprometimento gerencial. A CMM caracteriza essa prática.

\*REUSO COMPOSICIONAL

- Ocorre com a utlização de componentes existentes para a criação de um novo sistema, um exemplo é framework JSF.

\*REUSO BASEADO EM GERADORES DE CÓDIGOS

- Consiste no reuso no nível de especificação do sistema, usa-se geradores de códigos automáticos. Ex: Ferramentas UML.

\*REUSO CAIXA BRANCA

- Implementar de forma que o código consiga ser usado de maneira exposta, em exemplo disso é a herança em Java e C++. Desenvolvedor deve ter um pré conhecimento da implementação.

\*REUSO CAIXA PRETA

- Faz com que o desenvovedor não necessite ter um pré conhecimento do trecho a ser utilizado, esse tipo de reuso ocorre através da descrição de interfaces ou contratos bem definidos.

\*REUSO DE CÓDIGO FONTE

- É o mais utilizado na prática, que reusa o código já desenvolvido.

\*REUSO DE PROJETO

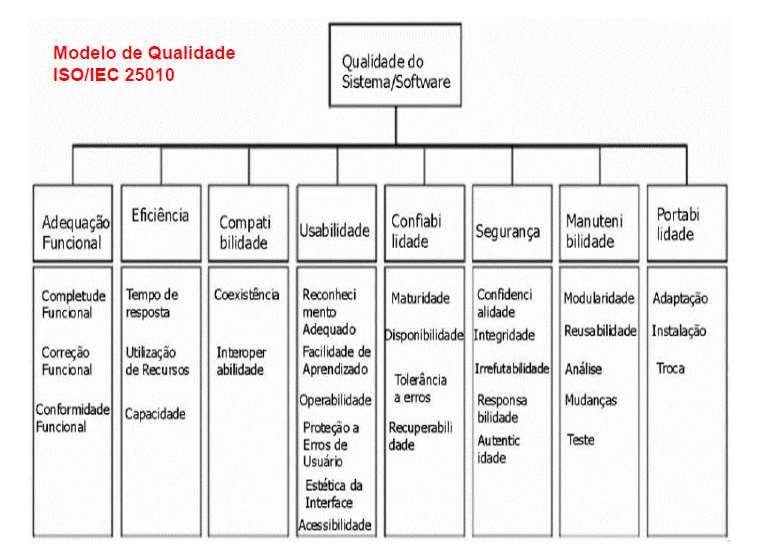
- Ofere um retorno maior que o reuso de código fonte, quanto maior o tamanho do componente a ser reusado, maior é o ganho em qualidade, desempenho, custo reduzidos, etc...

\*REUSO DE ESPECIFICAÇÃO

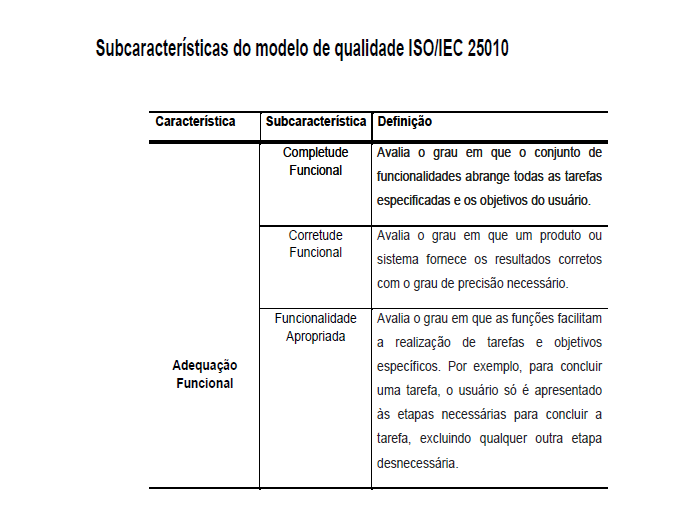
- Da mesma maneira que ocorre com projetos, quando se reutiliza uma especificação, tem- se, como conseqüência direta o reuso de projeto e do código fonte.

\*ARQUITETURA DE SOFTWARE

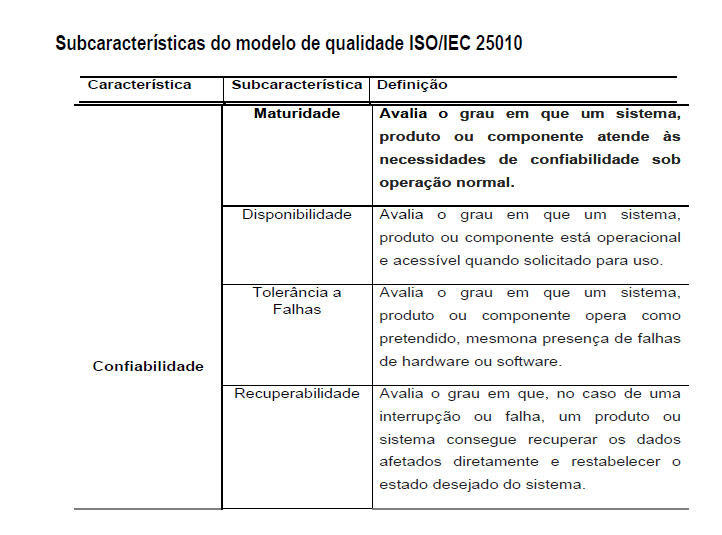
- Os softwares se tornaram os elementos chaves para o sucesso e bom desempenho dos negócios.

A qualidade de software está divida nos seguintes tópicos:

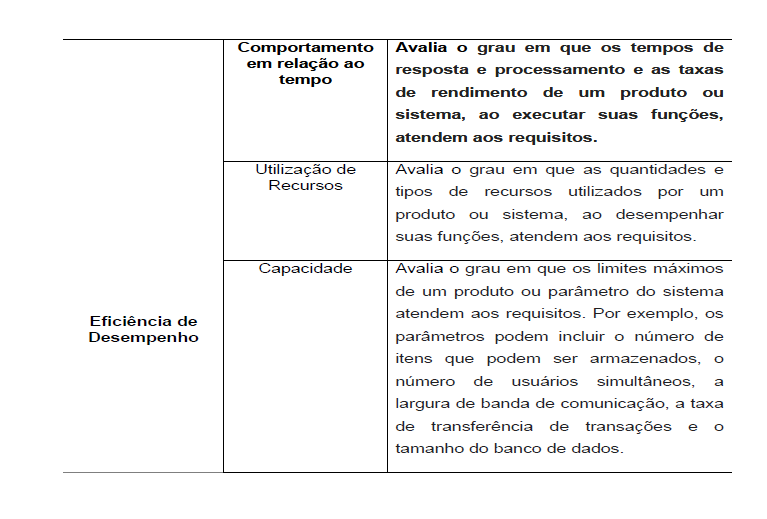
\*ADEQUAÇÃO FUNCIONAL:



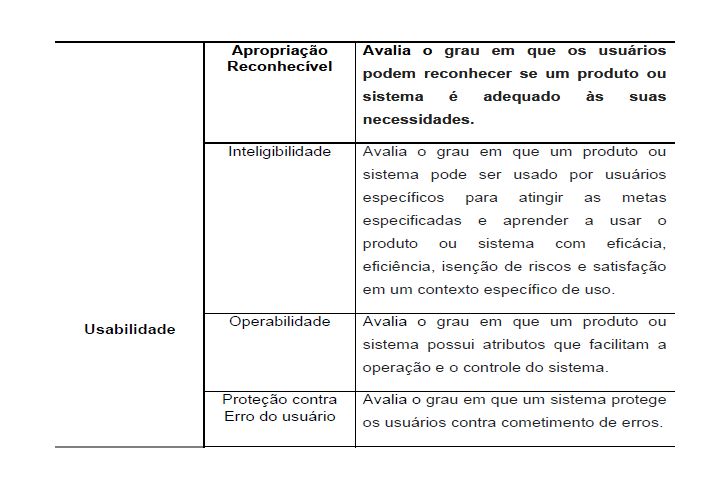
\*CONFIABILIDADE:

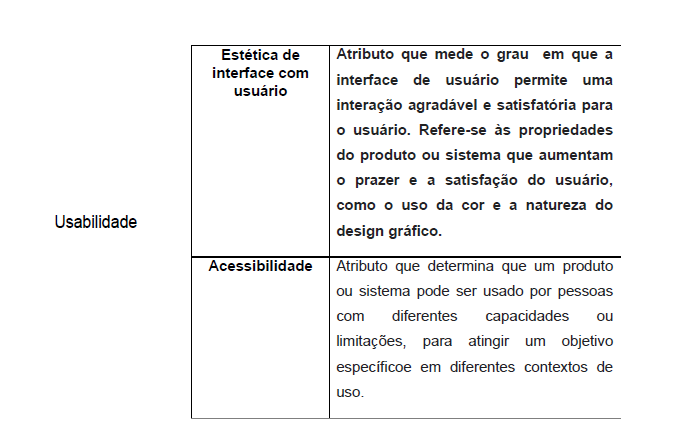


\*EFICIÊNCIA DE DESEMPENHO:

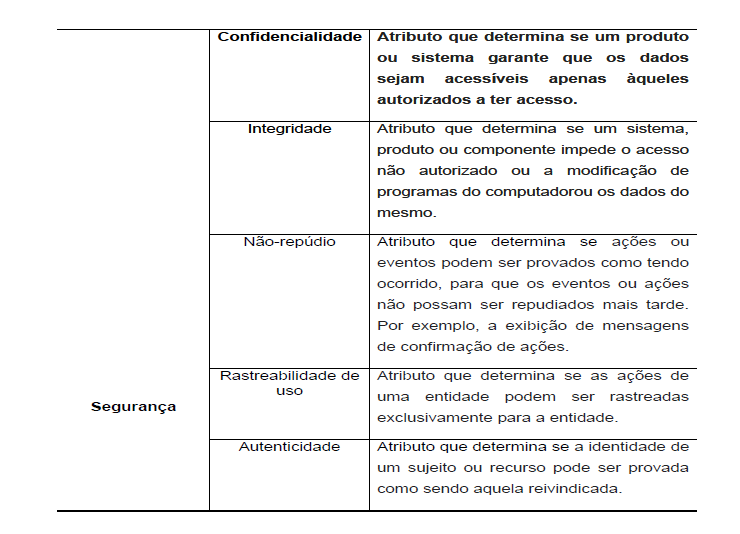


\*USABILIDADE:

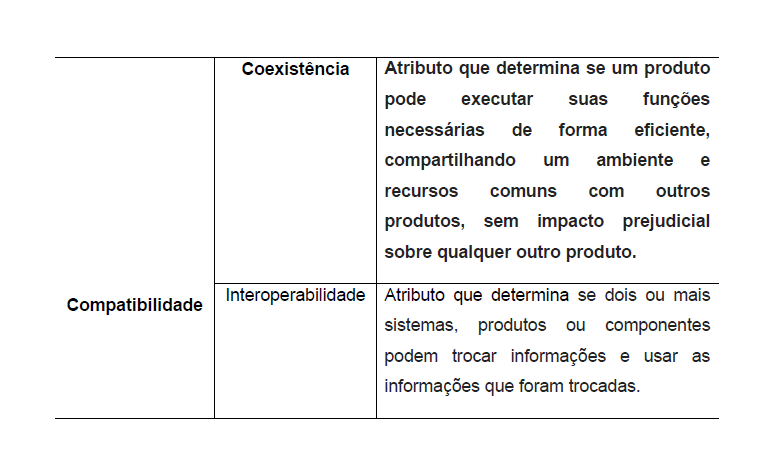




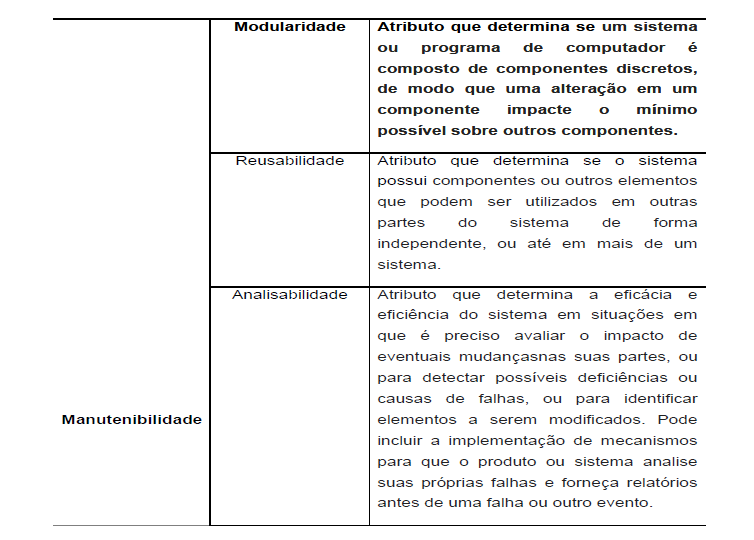
\*SEGURANÇA:

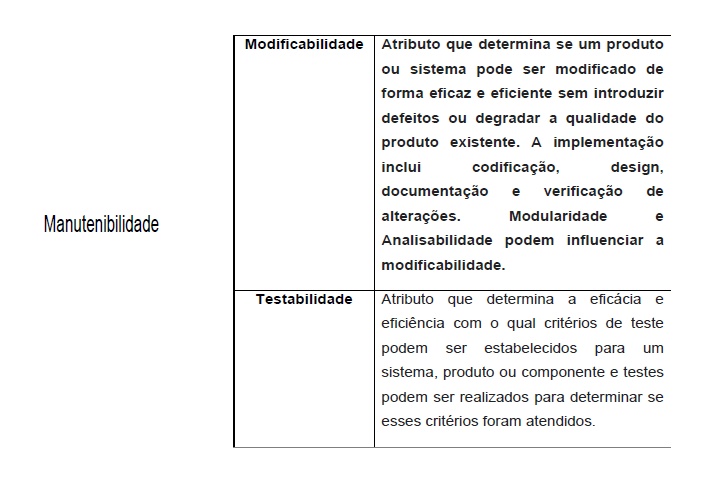


\*COMPATIBILIDADE:

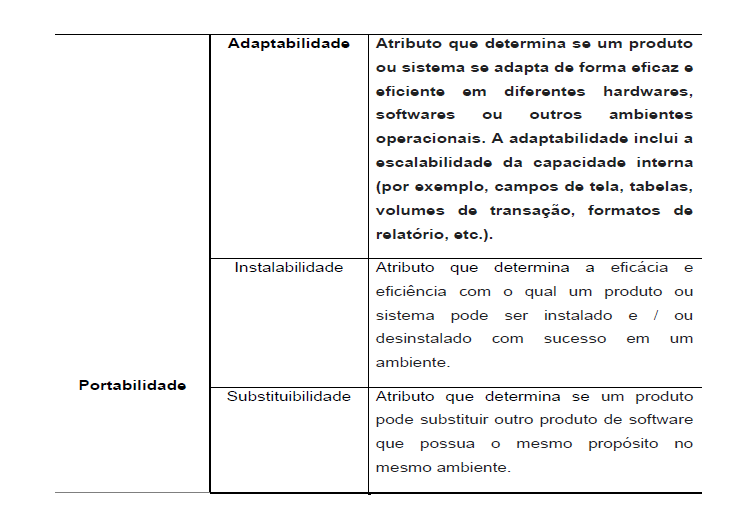


\*MANUTENIBILIDADE:





\*PORTABILIDADE:



\*DESENVOLVIMENTO DE SOFTWARE:

- CAUSAS DE INSUCESSO

- Iniciar o desenvolvimento se preocupando apenas com questões técnicas de desenvolvimento e infraestrutura e esquecer a documentação de análise e elicitação de requisitos.

- Utilizar novas tecnologias ainda não difundidas no mercado, ignorando o contexto, domínio e adequação do problema.

- Pouca flexibilidade para alterações não esperadas.

- Não focar em manutenção, confiabilidade e segurança.

- Para evitar esse trantorno surgiu o termo engenharia de software em 1968 durante a conferência da crise do software.

- ARQUITERTURA DE SOFTWARE:

- Trata-se de o esqueleto central e divisões de funções dos subsistemas e módulos. Ex: MVC

- É a estrutura central do sistema, através da organização de seus módulos.

- Processo de arquitetura de software:

Processo da arquitetura compreende nas atividades complementares que compõe a construção do software.

-Elaborar modelo de negócio;

-Entender os requisitos;

-Criação ou seleção de uma arquitetura;

-Representação da arquitetura;

-Implementar usando a arquitetura;

-Analisar/avaliar a arquitetura;

-Importância da Arquitetura:

-Organizar a estrutura de maneira geral;

-Definir protocolos de comunicação;

-Sincronismos;

-Atribuição de funcionalidade a componentes de projeto;

-Escalabilidade e desempenho;

-Seleção de alternativas;

-Reconhecer estruturas existentes para desenvolver novos sistemas;

-Permite aos engenheiros decisões mais precisas e novas alternativas;

-As restrições, regras de negócio e regras técnicas DEVEM ser atendidas pela arquitetura;

-A **arquitetura robusta** surge após extensa análise e ampla compreensão do projeto, a representação do mesmo torna-se robusta o suficiente para seguir a diante.

-Provém suporte ao reuso de software;

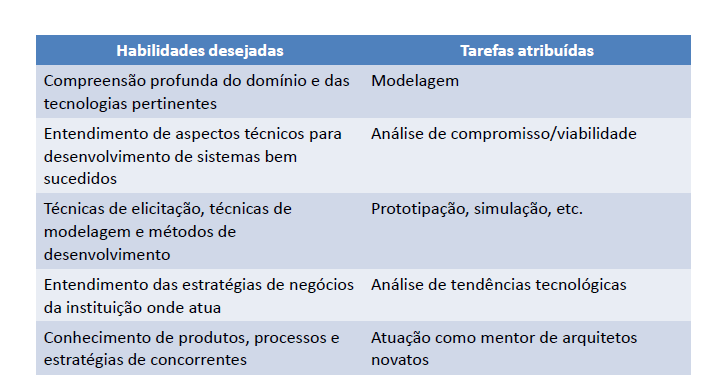
-Serve de base para análise da consistência e dependência;

\*O ARQUITETO

-Possui posição estratégica;

-Precisa saber sobre domínio do sistema, tecnologias relevantes e processo de desenvolvimento.

-O arquiteto deverá: construir modelos para o problema, achar soluções, gerar documentação.



-Ter facilidade de trabalhar em vários níveis de abstração;

-Ser criativo e investigar.

TÉCNICAS DE ARQUITETURA:

\*MODELO DE REFERÊNCIA

-Trata-se da decomposição padronizada do problema em partes conhecidas e que cooperam entre si para se atingir a solução. Este modelo surge durante o processo de amadurecimento das soluções devido a necessidade do domínio. Este modelo em especial habilita o desenvolvimento de arquiteturs específicas usando padrões consistentes. É constituído de um conjunto mínimo de conceitos da área.

Ex: para uma casa seria: a área de higiêne, refeição e etc... são conceitos/componentes que o compõe...

\*ESTILOS DE ARQUITETURA

-Cada estilo de arquitetura lida condiferentes tipos de atributos da qualidade, exemplos:

1. **Pipes e filtros**

POSSUI REDE ONDE OS DADOS ANDAM DE PONTA A PONTA, EXECUÇÃO EM PARTES/LOTES...

A arquitetura de pipes e filtros consiste na construção de um programa assumindo-se que dados devem fluir de um ponto a outro por meio de canos (pipes), passando, durante esse processo, por meio de filtros, que vão realizando o processamento dos dados em passos. Esse tipo de sistema é facilmente entendível, mantido e reutilizado, além de apresentar um desempenho maior caso implemente-se processamento paralelo.

Essa arquitetura, entretanto, é indicada para projetos de sistemas mais complexos, que exigiriam diversas etapas de processamento. Usá-la em um sistema em que as ações realizadas são deveras simples (como, por exemplo, consultar um livro), pode acabar sendo a decisão errada, pois não aproveitaria ao máximo a capacidade da arquitetura.

Além disso, essa forma de desenvolvimento foca no processamento no modo batch (sem grande interação com o usuário), o que significa que sistemas que necessitam de um maior nível de interatividade (tais como sistemas de biblioteca), são comprometidos pela forma como o sistema processa os dados. Por esse motivo, esta arquitetura foi descartada pela equipe.

Vantagens: As vantagens de utilizar o modelo arquitetural de Pipes e Filtros é que estes podem ser vistos como caixas pretas, onde há [encapsulamento](https://pt.wikipedia.org/wiki/Encapsulamento), alta coesão, recombinação e reuso dos dados. Isso implica afirmar que esse estilo tem suporte à reutilização. Além disso, os filtros interagem com os outros componentes de forma limitada - o que leva a um baixo acoplamento. O mais relevante é que o sistema pode ser facilmente estendido e modificado, facilitando a implementação em processadores paralelos ou em múltiplas [threads](https://pt.wikipedia.org/wiki/Threads) em processadores simples.

Desvantagens: As desvantagens de utilizar esse estilo arquitetural é que, devido o processamento ocorrer em lotes, é difícil criar aplicações interativas. Geralmente filtros exigem que os dados sejam representados no denominador comum mais baixo, tipicamente fluxos de bytes ou caracteres. Esse estilo arquitetural também pode introduzir sobrecarga para analisar o fluxo de dados, podendo haver a exigência de um [buffer](https://pt.wikipedia.org/wiki/Buffer) de tamanho limitado, que devido à essa limitação, pode causar um [deadlock](https://pt.wikipedia.org/wiki/Deadlock). Todas essas desvantagens geram uma baixa performance.

1. **Camadas**

AGRUPA UM TIPO DE TAREFAS, CAMADAS SUPERIORES USAM SERVIÇOS DE CAMADAS INFERIORES, EM CASOS DE MANUTENÇÕES DEVEMOS NOS PREOCUPAR COM CAMADAS SUPERIORES, SUPORTE A PORTABILIDADE, MAIS CAMADA=MAIS LENTO

Esse estilo arquitetural pode ser melhor entendido usando-se conceitos de redes de computadores. Como no modelo OSI, as funcionalidades do sistema são agrupadas, por nível de abstração, em diversas camadas. Essas camadas comunicam-se por meio de uma interface, mas ignoram o que exatamente acontece dentro das outras camadas. Isso significa que sistemas nessa arquitetura são mais fáceis de alterar, uma vez que não geram muito backtracking.

No sistema da biblioteca, a vantagem do uso da arquitetura em camadas seria a facilidade com a portabilidade, o que facilitaria a transformação do website em um aplicativo em projetos futuros, por exemplo. Essa arquitetura, entretanto, é recomendada para sistemas muito complexos, que necessitam deste tipo de divisão, uma vez que a separação em camadas gera custos altos em desempenho. Para um sistema como o da biblioteca, que precisa que de apenas alguns sistemas mais simples, um sistema em camadas seria um desperdício de arquitetura. Por esse motivo, essa arquitetura foi descartada pela equipe.

Vantagens: Separação de código relativo a interface com o usuário (UI), comunicação, negócio e dados. Permite a mudança de implementação de uma camada sem afetar a outra, desde que a interface entre as mesmas seja mantida. Possibilita que uma camada trabalhe com diferentes versões de outra camada.

Desvantagem: Aumento significante no número de classes existentes no sistema

1. **Invocação implícita**

A arquitetura baseada em invocação implícita está diretamente ligada a paradigmas de programação orientada a eventos. Nesse tipo de arquitetura, ao invés de funções serem invocadas para realizar alguma atividade, um agente dispara um evento informando alguma mudança no sistema. Outros agentes que queiram fazer algo com a informação devem ter um listener, para ouvir quando o evento for disparado, e então processar a informação. Sistemas interativos tendem a possuir este tipo de característica. Num sistema de pedido de comida, por exemplo, quando o usuário finaliza o pedido, um evento anunciando um novo pedido é disparado, então, o restaurante é notificado da chegada deste pedido. Essa arquitetura, entretanto, tem um contratempo. Quando um evento é disparado, o componente que o disparou não sabe se será respondido, e nem em que ordem. No sistema da biblioteca, por exemplo, se o cliente fazer um reserva online fosse um evento, não é possível saber se quem responderia primeiro seria o ”reservar livro” ou retirar item do estoque, e a flutuação da ordem de execução¸ desses passos poderia ocasionar inconsistências e problemas. Além disso, o paradigma orientado a eventos é menos simples de compreender, uma vez que os membros do time aprenderam outros paradigmas mais profundamente, o que tornaria o desenvolvimento do sistema mais complexo e não significativamente melhor. Por esses motivos, essa arquitetura foi descartada pela equipe.

Vantagens: Forte suporte ao reuso e facilita a evolução do sistema.

Desvantagens: Um componente que anuncia um evento não tem controle (ordem/tempo) sobre a execução associada, a troca de dados precisa ocorrer também de forma indireta (repositório compartilhado).

1. **Objetos**

O.O, TROCAS DE MENSAGENS, OBJETOS QUE RELACIONAM ENTRE SI

A arquitetura orientada a objetos é uma das mais difundidas atualmente. Ela considera unidades de complexidade como objetos (na biblioteca, por exemplo, bibliotecário, balconista, livro). Esses objetos possuem características (idade, peso, validade, etc.) e também ações (bibliotecário cadastra o livro, por exemplo).

Esse paradigma é base pra diversas linguagens muito populares atualmente, tais como Java e C++. Além disso, a implementação de um sistema de bancos de dados é mais intuitiva partindo-se dessa arquitetura, uma vez que tabelas, por vezes, podem ser representadas como Classes. Uma vez que o sistema deverá ter um banco de dados para armazenar informações acerca de produtos, vendas, pagamentos e etc., a arquitetura orientada a objetos é vista como a melhor opção.

Além disso, existe uma base literária maior para sistemas de gerenciamento desenvolvidos usando essa arquitetura, o que facilita o trabalho dos membros da equipe quando decidirem buscar base teórica para alguma tomada de decisão de projeto.

Por fim, uma vez que grande parte do curso é construída sobre conhecimentos de orientação a objeto, as linguagens desse paradigma são simples e poderosas, além de dar suporte para versões web, e os membros da equipe se sentem mais confortáveis com elas, decidiu-se que a arquitetura utilizada para a implementação do sistema será a orientada a objetos.

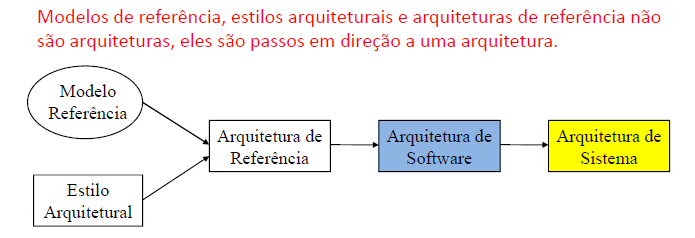
Vantagens:  Reusabilidade de código, Escalabilidade de aplicações, Manutenabilidade, Apropriação.

Desvantagens:  Apropriação, Fragilidade, Linearidade de desenvolvimento.

\*ARQUITETURA DE REFERÊNCIA:

-Consiste nos componentes e os relacionamento entre eles para se formar um determinada funcionalidade, são aplicáveis a um domínio particular, ex: banheiro, cozinha, quarto etc...

Obs: O conceiro de área de refeição é um modelo de referência, já a cozinha e a realização da área de refeição... São coisas distintas...



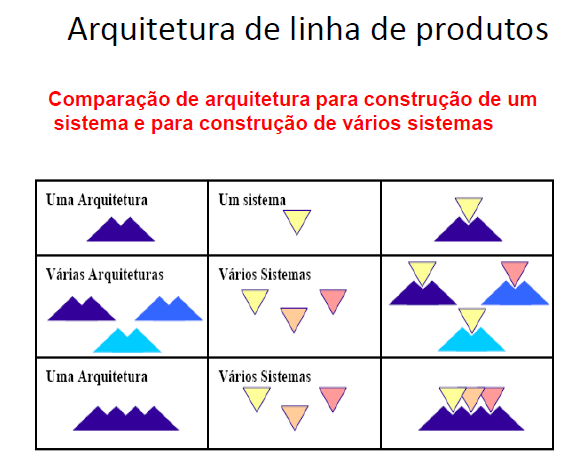
A arquitetura de referência apresenta soluções técnicas mas se baseando nas soluções de negócio com base no modelo de referência.

\*ARQUITETURA DE LINHA DE PRODUTOS

-Define conceitos, estruturas, componentes e restrições.

-Um sistema pode ter vários produtos que compõe uma linha de produtos com características semelhantes.

-A linha de produtos podem compartilhar a mesma arquitetura.



ESTILOS ARQUITETURAIS

Note que o estilo arquitetural mais apropriado para um sistema, depende de seus requisitos funcionais e não funcionais.

\*PIPES E FILTROS:

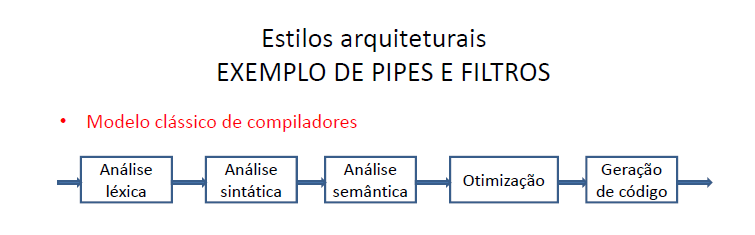
-Exige rede pela qual os dados fluem;

-Fluxo de dados através de dutos e sofrem mudanças quando filtrados;

-Um pipe provê uma forma unidirecional para fluidez dos dados até o destino;

-Uma sequencia de pipes forma tubulação de processamento;

-Adequada para um projeto de varios estágios de processamentos.



\*VANTAGENS

-Problemas podem ser decompostos de forma hierarquica;

-Facilita entendimento;

-Manutenibilidade flexível;

-Alto reuso...

\*DESVANTAGENS

-Pode haver deadlock;

-Mudanças frequentes podem reduzir a manutenibilidade.

\*CAMADAS

-Conjunto de camadas, onde cada uma delas agrupam tarefas a um determinado nível de abstração;

-Não há especificação quanto a granularidade de componentes;

-Maior flexibilidade;

-Suporte aportabilidade, já que é constituída de camadas;

-Quando a interface da camada é alterado, surgem novos recursos;

-Apoia o desenvolvimento incremental de sistemas;

-DETERMINAR NÚMERO ADEQUADO DE CAMADAS E DIFÍCIL.

\*OBJETOS

-Tem como base o uso de um tipo abstrato: o objeto, que são entidades que podem sofrer modificações, já que suas informações são mantidas em si mesmos.

-É possível fazer um mapeamento dos objetos de modo a compreender melhor o sistema.

-Podem ser reutilizados e fazem trocas de mensagens.

-O sistema é um conjunto de objetos comunicantes.

\*INVOCAÇÃO IMPLÍCITA

-Um componente pode anunciar um ou mais eventos;

-Um evento sendo anunciado, causa implicitamente a invocação de procedimentos.

-Fornece suporte para reuso;

-Facilita a evolução do sistema;

\*SISTEMAS ORIENTADOS A EVENTOS

-São regidos por um estimulo externo, um exemplo seria um sinal recebido de outro dispositivo/sensor.

-Os sistemas orientados a interrupção são usados nos sistemas de tempo real, no qual as interrupções são detectadas e tratadas...

MODELO BROADCAST:

Os subsistemas registram interesses a eventos, no qual o tratador de evento envia para o controlador correspondente para aquele determinado tipo de evento.

-Possui evolução simples, qualquer subsistema pode ativar outro subsistema, podem ser implementados em máquinas distribuídas.

\*QUADRO-NEGRO

Originado na inteligencia artificial, espera-se que haja uma base de dados central e é usada para processar dados em busca de soluções de problemas/conhecimento. Vários agente cooperam entre si para resolver o problema.

Um ou mais componentes interagem com o banco de dados compartilhado em busca da solução do problema, envolve um único reposítório central de dados, mas em casos de problemas muito complexos, mais repositórios podem ser usados.

Dá suporte a manutenibilidade, desempenho nem tão bom, pois a célula de conhecimento deve passar pelo quadro-negro.

\*APLICAÇÕES DISTRIBUÍDAS

Presença de vários processadores, podendo ser multiprocessador ou multicomputador

Filtros: transformador de dado;

Clientes: inicia uma requisição;

Servidor: recebe a requisição e retorna resultado;

CLIENTE-SERVIDOR:

-Servidor é reativo, fica esperando uma requisição do cliente;

-Cliente é conectado ao servidor via rede, é ativo e realiza as solicitações ao servidor;

PONTO A PONTO P2P

-A computação de dados pode ser realizada por qualquer nó da rede, servidor e clientes não possuem distinção, criado com intuito de usar a capacidade computacional e de armazenamento a nivel de rede mundial.

-Altamente resuntante, tolerante a nós que se desconectam da rede.

\*SISTEMA ORIENTADO A SERVIÇOS

Empresas podem deixar suas informações acessíveis a outros programas através da interface de web service, onde é definido os dados disponíveis e como podem ser acessados.

O fornecimento de serviço é independente da aplição que usa o serviço.

SISTEMAS DE INFORMAÇÃO

É uma classe de sistemas na qual há armazenamento, recuperação e processamento de dados, ou seja, uso de banco de dados.

Há quem diga que o banco de dados é a principal parte dessa estrutura.

Um SI deve ter: desempenho, segurança, disponibilidade, integridade e usabilidade.

SIST. INF COM CLIENTE-SERVIDOR:

Principais componentes, são: rede, cliente e servidor;

CLIENTE SERVIDOR COM DUAS CAMADAS

CLIENTE MAGRO

-Todo processamento é realizado no servidor, o usuario apenas usa a interface de apresentação das informações.

-Mais simples e de fácil manutenção, mas pode sobrecarregar o servidor.

CLIENTE GORDO:

-O servidor é um gerenciador de dados, o cliente implementa a lógica e as interações com usuários.

-Distribui melhor o processamento de dados, mas a complexidade deste tipo de sistema aumenta.

CLIENTE SERVIDOR COM MÚLTIPLAS CAMADAS

-Surgiram os servidores de aplicação, no qual se encarregam de ter a parte lógica da aplicação e faz a mediação entre cliente e recursos.

-Trouxe melhorias para as dificuldades entradas na arquitetura de duas camadas, aumentou o desempenho.

-Mais escalonável e o tráfego de rede foi reduzido.

CLIENTE SERVIDOR COM MÚLTIPLAS CAMADAS PRA **WEB**

-Redesenhou os negócios em virtude da infra estrutura.

-Informações acessíveis através de dispositivos com conexão a rede.

-Baixo custo de manutenção;

-Acesso de qualquer lugar, etc...

SISTEMAS DE TEMPO REAL

Deve responder aos estimulos em pequeno intervalo de tempo, depende dos resultados obtidos e do tempo que levou para se obter os resultados.

SISTEMA DE TEMPO REAL LEVE:

A operação e degradada caso o resultado não venha no timing esperado.

SISTEMA DE TEMPO REAL RÍDIDO:

A operação será incorreta se os resultados demorarem a chegar.

Esse tipo de sistema deve ter uma arquitetura organizacional de modo que os estímulos recebidos sejam encaminhados ao tratador correto o quanto antes.

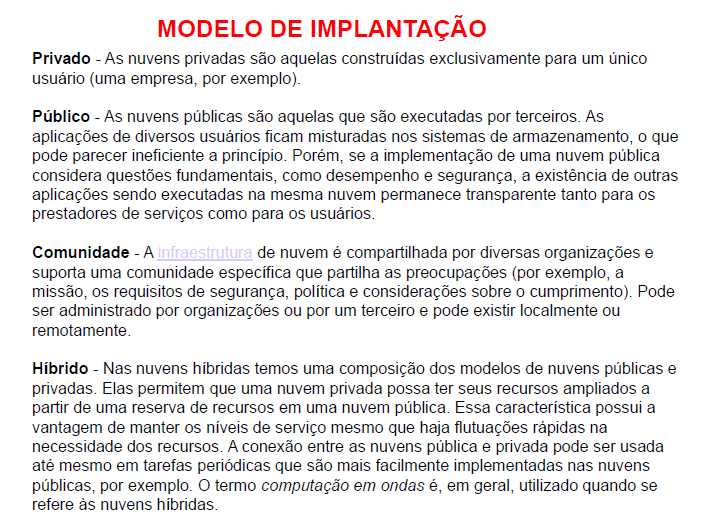
Gerenciamento de sensores, computacionais e atuadores.

Possível criar códigos EFICIENTES a partir de linguagens de programação de baixo nível: C e Assembler

ARQUITETURA EM NUVEM

O acesso ao sistema, funcionalidades, serviços e dados ocorrem a partir da internet.

Possível usar softwares sem que sejam instalados em seu computador, mas se ficar em internet, tudo ficará indisponível ate que ela volte.



**MVC**

-Surgiu com uso intensivo de frameworks a partir da década de 70.

-Paradigma de desenvolvimento e design que divide a aplicação em 3 partes: MODEL, VIEW E CONTROLER.

-Faz a separação da lógica de negócio da entrada de dados/apresentação, possui reusabilidade e reduz os esforços na manutenção do software.

VIEW

Responsável pela parte gráfica, apresentação de dados e textos.

Se liga ao model e renderiza seu conteúdo, pode haver várias visões para um mesmo modelo

MODEL

Gerencia os dados, responde a consultas quando chamado, atualiza os dados e possui as regras de negócio alí.

Contém dados e funcionalidades pertinentes aos dados envolvidos

CONTROLLER

Interpreta as entradas e chama as ações/métodos do model para manipular os dados e realizar a mudança de estado.

O MVC visa atingir maior independência entre o M e o V, não importa de onde vem os dados, a view deve garantir que serão apresentados da mesma forma que foi enviado pela model. O model não quer saber qual botão ou ação no front end o requisitou e sim em alterar oe stado corretamente.

Aceita a entrada de dados e instrui o model e view sobre o comportamento esperado e mduanças de estados que devem ocorrer e ser apresentadas na view.

Seleciona a view correspondente.

**MVC X 3 CAMADAS**

MVC é um padrão de arquitetura que atua na GUI (Graphical User Interface)

A camada de apresentação nunca acessa os dados diretamente

Não há uma relação típica de camadas, onde uma camada depende da outra exclusivamente da outra.

A interação é mais complexas e envolve trocas de mensagens...

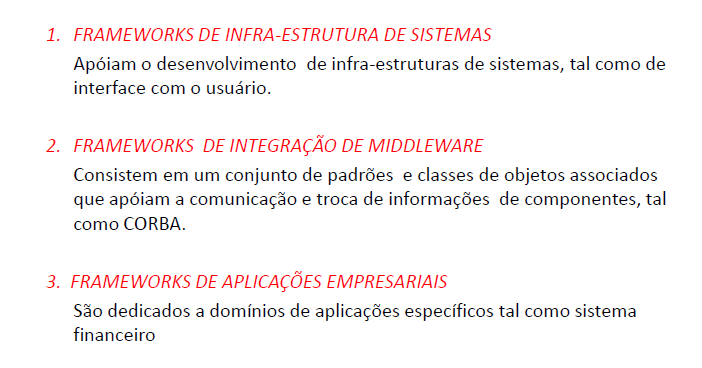
Ex: JAVA, HTML, PHP, C#, etc..

Modelo quem faz todo o trabalho...

**FRAMEWORKS**

O reuso pode ocorrer muito melhor em um ambiente de O.O com a utilização de framworks, que são um subsistema composto por classes abstratas e interfaces entre elas.

Frameworks fornecem a solução inicial para um problema muito extenso, fazendo acom que ganhe tempo para encontrar a solução.

Eleva a produtividade, menos bugs, aplicações mais homogênes

Um frameworks vem para resolver e fornecer mecanismos para solução de problemas comuns em várias aplicações, só é possível criar um framework quando há um mesmo domínio de problema e a intersecção é grande.

Provê soluções para uma grande gama de problemas, as classes e interfaces decompõe a família de problemas, cada classe colabora para resolversua parte do problema, conjunto de classes deve ser flexivel e extensivel.

Frameworks são uma peça incompleta, você que os deve completar.

Você deve se adaptar ao modelo especificado pelo framework

