INF016 – Arquitetura de Software

03 – Conceitos Básicos

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Objetivos

- Definição dos principais termos e ideias da área de arquitetura de *software*
- Realizar um nivelamento uniforme para o prosseguimento do curso
- Veremos definições e exemplos de:
  - Principais elementos de uma arquitetura e seus relacionamentos
  - Técnicas e processos básicos para desenvolver uma arquitetura de *software*
  - Principais *stakeholders* e os papéis por eles desempenhados no processo

Arquitetura de Software

- Na sua essência a arquitetura de um *software* pode ser definida como:

Arquitetura de Software: conjunto formado pelas principais decisões de projeto tomadas a respeito do sistema

Arquitetura de Referência: conjunto formado pelas principais decisões de projeto que são simultaneamente aplicadas a múltiplos sistemas relacionados, geralmente dentro de um domínio de aplicação, com pontos de variação explicitamente definidos

Arquitetura de Software

- Mas o que é uma “decisão de projeto” ? Exemplos:
  - Relacionada à estrutura do sistema: “os elementos arquiteturais devem ser organizados e compostos da seguinte forma: ...”
  - Relacionada ao comportamento funcional: “o processamento, armazenamento e visualização de dados serão realizados exatamente nesta sequência”
  - Relacionada a interações: “a comunicação entre todos os elementos do sistema será realizada somente através de notificação de eventos”
  - Relacionada a propriedades não-funcionais: “*dependability* será garantida através de módulos replicados de processamento”
  - Relacionada à implementação do sistema: será utilizado o *Java Swing* para os componentes de *GUI*

Arquitetura de Software

- Nem todas as decisões de projeto são “principais”:
  - Depende do grau de importância e particularidade da decisão
  - Detalhes dos algoritmos e estruturas de dados utilizados não são decisões principais
  - Pode depender das metas do sistema e dos interesses dos *stakeholders*
- Resumindo, a arquitetura é também determinada pelo contexto (eventualmente dirigido por questões não-técnicas)
- Diferentes *stakeholders* podem julgar como principais diferentes conjuntos de decisões

Arquitetura de Software

- Todo conjunto de decisões principais de projeto pode ser visto como uma arquitetura diferente
- Ao longo da vida do sistema, as decisões serão tomadas, descartadas, evoluídas, bifurcadas e convergidas
- O resultado é um conjunto de centenas de arquiteturas diferentes porém relacionadas
- A arquitetura de *software* tem, portanto, um aspecto temporal

Arquitetura Prescritiva

- Arquitetura prescritiva:

Arquitetura Prescritiva: conjunto *P* formado pelas principais decisões arquiteturais tomadas pelos arquitetos em um tempo *t* qualquer. É a prescrição para a construção do sistema

  - Representa a arquitetura “pretendida” ou “concebida” do sistema
  - Pode não existir de forma tangível, apenas na cabeça do arquiteto ...
  - … ou pode ser capturada através de alguma notação ou outra forma de documentação

Arquitetura Prescritiva

- As decisões que compõem a arquitetura prescritiva serão implementadas por um conjunto *A* de artefatos:
  - Representação das decisões arquiteturais em *UML*
  - Implementações em uma linguagem de programação
  - Modelos dos estilos arquiteturais e padrões utilizados
  - Componentes *COTS* a serem utilizados
  - Infra-estruturas de *middleware* e *frameworks*
- Cada artefato em *A* encapsula certas decisões de projeto

Arquitetura Descritiva

- Arquitetura descritiva:

Arquitetura Descritiva: conjunto *D* formado pelas principais decisões arquiteturais encapsuladas por todos os artefatos do conjunto *A*. Descreve como o sistema foi implementado

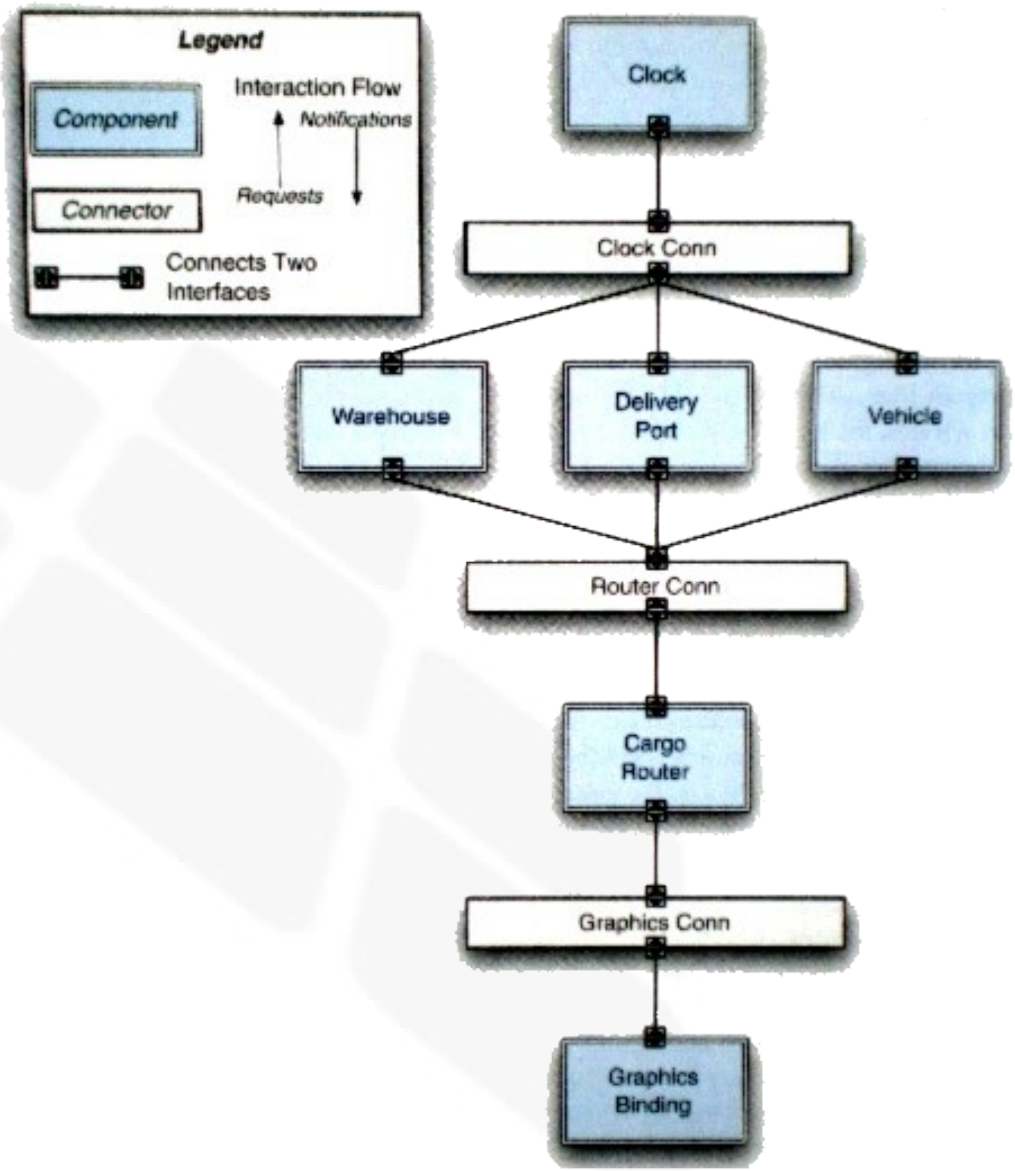
  - Representa a arquitetura “implementada” do sistema
  - No início do projeto (tempo t_1) tem-se a criação do conjunto P_1 e os conjuntos A_1 e D_1 podem estar vazios (desenvolvimento *greenfield*)
  - No desenvolvimento *brownfield*, onde um conjunto de artefatos implementando parcialmente a arquitetura já existe, A_0 e D_0 são não-vazios enquanto P_0 é vazio

Arquitetura Descritiva

- P_0 também pode estar vazio e D_0 com alta cardinalidade em um projeto envolvendo um sistema legado cuja intenção arquitetural foi perdida ao longo do tempo
- Tais discrepâncias entre os conjuntos *P* e *D* podem ser indícios de problemas na arquitetura do sistema

Arquitetura Prescritiva e Descritiva

- Exemplo: arquitetura prescritiva

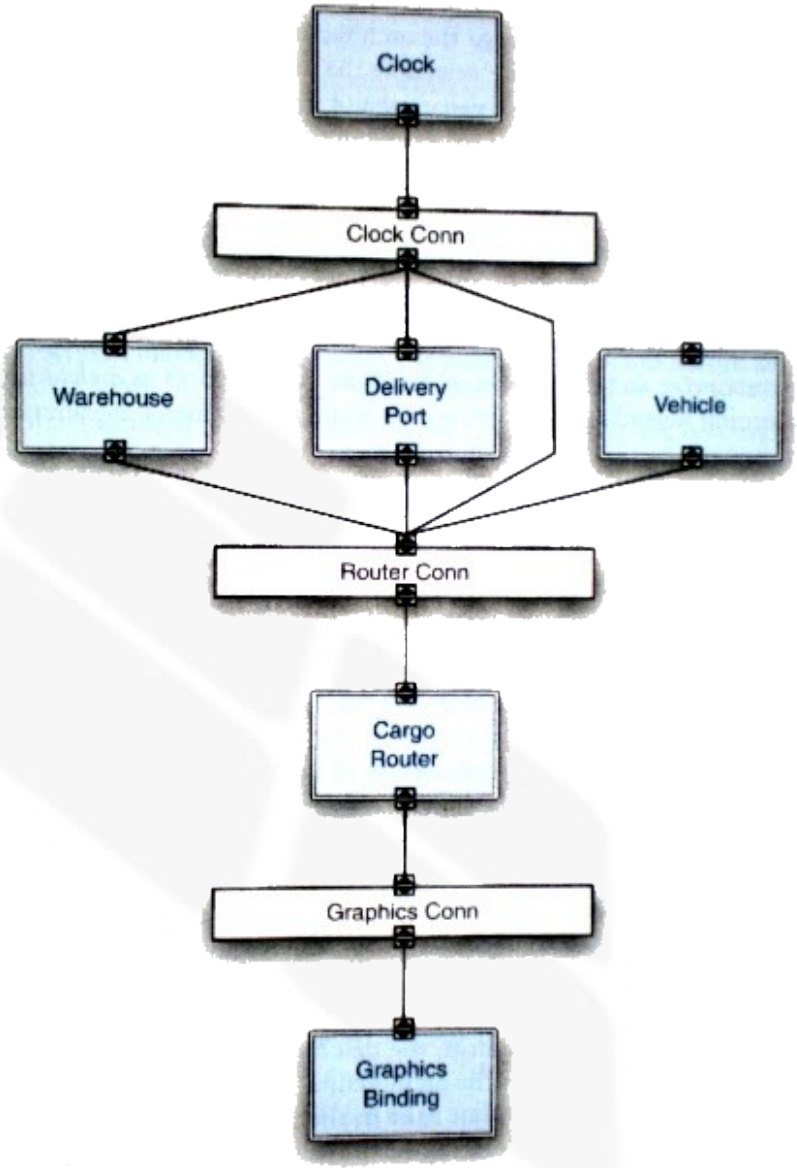


Arquitetura Prescritiva e Descritiva

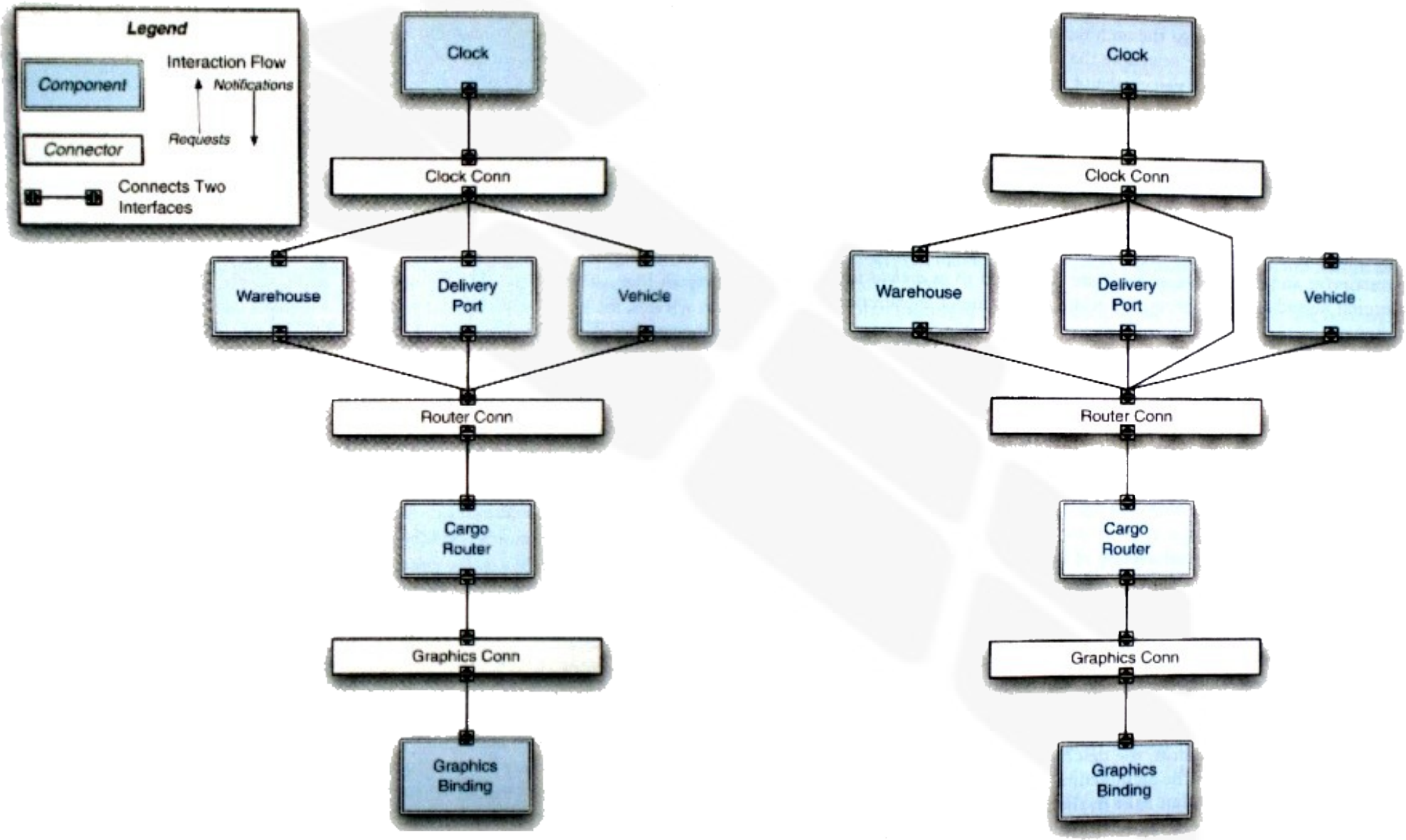
- O exemplo é útil porque:
  - Sua simplicidade sugere que seus arquitetos puderam avaliar todas as decisões arquiteturais e seus *trade-offs* antes da implementação da aplicação
  - A arquitetura foi implementada com a ajuda de um *architectural framework*, permitindo implementar todas as decisões diretamente em código
  - Somente a biblioteca utilizada na *GUI* é *COTS*, o que permitiu um maior controle do mapeamento entre a arquitetura e a aplicação

Arquitetura Prescritiva e Descritiva

- Exemplo: arquitetura descritiva



Arquitetura Prescritiva e Descritiva



Arquitetura Prescritiva e Descritiva

- Questões:
  - Qual arquitetura é a “correta” ?
  - As duas arquiteturas são consistentes uma com a outra ?
  - Quais critérios foram utilizados para estabelecer a consistência entre as duas arquiteturas ?
  - As respostas às questões anteriores são baseadas em que informação ?

Degradação Arquitetural

- Degradação Arquitetural:
  - Durante a vida de um sistema várias arquiteturas prescritivas e descritivas serão criadas
  - Cada par correspondente de arquiteturas representa o sistema em um determinado tempo *t*
  - Quando os conjuntos *P*, *A* e *D* estiverem suficientemente completos e consistentes a aplicação é implantada
  - Em um cenário ideal *P* será sempre igual a *D*
  - Nem sempre é o caso:
    - *COTS* ou plataformas de *middleware* podem interferir nas decisões arquiteturais tomadas
    - É preciso que os *stakeholders* definam o limite aceitável de diferenças entre *P* e *D*

Degradação Arquitetural

- Degradação Arquitetural:
  - É possível que, dado os conjuntos *P* e *D* no tempo *t*, esses conjuntos permaneçam estáveis no tempo *t+1*, mesmo com um crescimento de *A*
  - É também possível que *P* mude enquanto *D* permanece o mesmo
  - De forma similar, *D* pode mudar enquanto *P* permanece o mesmo

Degradação Arquitetural

- Degradação Arquitetural:
  - Durante uma evolução o ideal é alterar primeiro *P* e depois *D*
  - Nem sempre isso acontece:
    - Por desleixo do desenvolvedor
    - Por prazos curtos que impedem o raciocínio e a documentação do impacto na arquitetura prescritiva
    - Por ausência de documentação da arquitetura prescritiva
    - Necessidade ou desejo de otimizar o sistema, “fato que pode ser feito somente no código”
    - Técnicas e ferramentas inadequadas
  - Qualquer que seja a razão elas são falhas e potencialmente perigosas

Degradação Arquitetural

Degradação Arquitetural: discrepância existente entre as arquiteturas prescritiva e descritiva do sistema

Desvio Arquitetural: é a introdução, na arquitetura descritiva do sistema, de decisões principais de projeto que: *a*) não estão incluidas na arquitetura prescritiva ou não são implicações dela, mas *b*) não violam nenhuma das decisões de projeto da arquitetura prescritiva

Erosão Arquitetural: é a introdução, na arquitetura descritiva do sistema, de decisões principais de projeto que violam decisões da arquitetura prescritiva

Degradação Arquitetural (Desvio)

- O desvio arquitetural é resultado de mudanças no conjunto *A* que resultam, por sua vez, em mudanças no conjunto *D*
- Nem todas as expansões de *D* resultam em desvio arquitetural:
  - Ex: *P* requer que criptografia seja utilizada na comunicação em rede pública e *D* pode afirmar que um algoritmo de chave pública será utilizado para suportar tal comunicação
- Exemplo de desvio arquitetural: ligação entre dois conectores na figura anterior:
  - Não existe em *P*, porém não foi afirmado que ligações entre conectores não poderiam ser realizadas

Degradação Arquitetural (Desvio)

- Desvios arquiteturais podem causar violações nas regras do estilo arquitetural
- Refletem a insensibilidade do engenheiro em relação à arquitetura do sistema, podendo conduzir a perdas na claridade da forma e da compreensão do sistema
- Se não apropriadamente corrigidos, desvios arquiteturais frequentemente evoluem para erosões arquiteturais

Degradação Arquitetural (Erosão)

- Erosões arquiteturais produzem sistemas difíceis de entender e adaptar e frequentemente com falhas em potencial
- Podem ocorrer quando um sistema sofre vários desvios e as decisões estão obscurecidas por várias mudanças pequenas intermediárias
- Embora seja menos provável de acontecer e mais fácil de corrigir, uma arquitetura pode sofrer erosão ser ter tido desvios anteriores
- Pode ser causada por decisões que funcionam bem isoladas mas geram problemas em conjunto

Degradação Arquitetural (Erosão)

- Exemplo de erosão arquitetural: remoção da ligação entre o componente *Vehicle* e o conector *ClockConn*, se ela não for justificada e presente também na arquitetura prescritiva:
  - Veículos controlados pelo sistema podem não sincronizar satisfatoriamente com os outros elementos
  - Caso tenha sido observado que os veículos conseguem sincronizar apenas com informações obtidas do componente *CargoRouter* então isto seria discutido com os arquitetos e a arquitetura prescritiva atualizada. Não teríamos erosão neste caso
- Tanto desvio quanto erosão arquitetural são perigosos e devem ser evitados

Perspectivas Arquiteturais

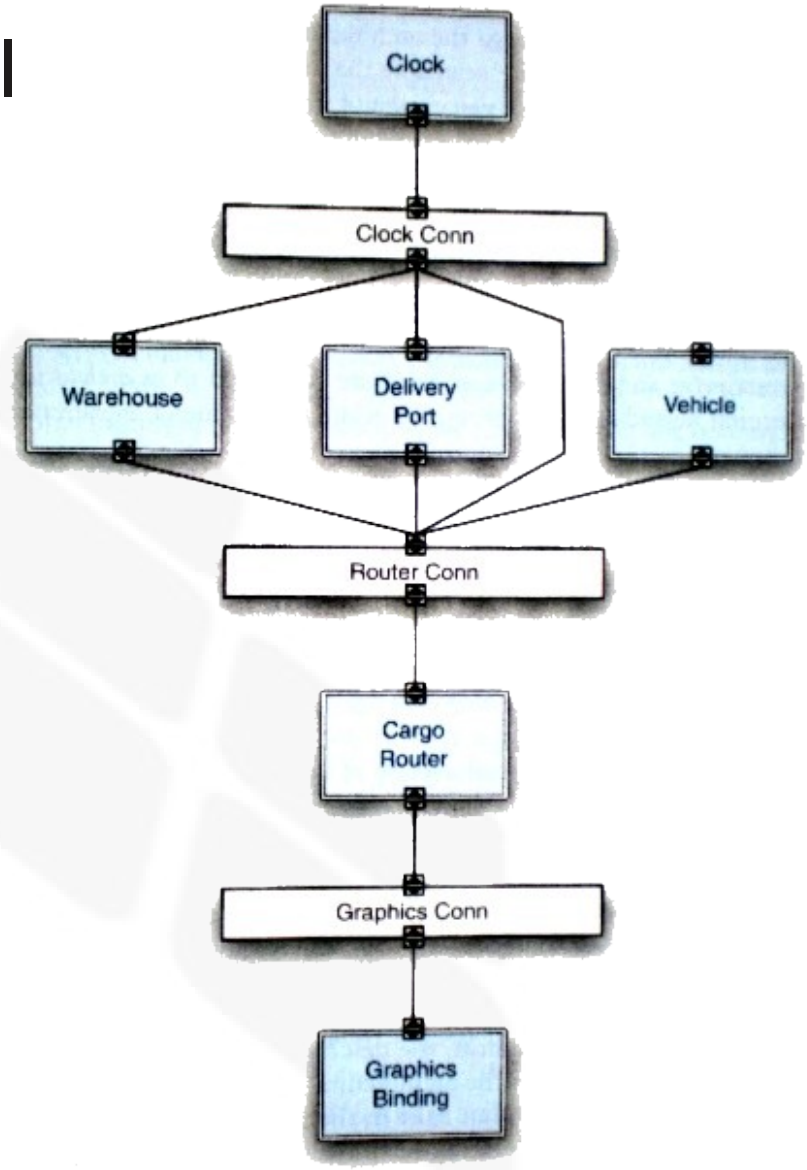
- Perspectiva (visão) arquitetural:
  - Tem como objetivo destacar determinados aspectos de uma arquitetura ao mesmo tempo em que omite outros

Perspectiva Arquitetural: é um conjunto não-vazio de tipos de decisões arquiteturais de projeto

  - Diversos *stakeholders* acrescentam decisões em diferentes detalhes e níveis de abstração
  - Uma perspectiva arquitetural direciona a atenção a um subconjunto dessas decisões

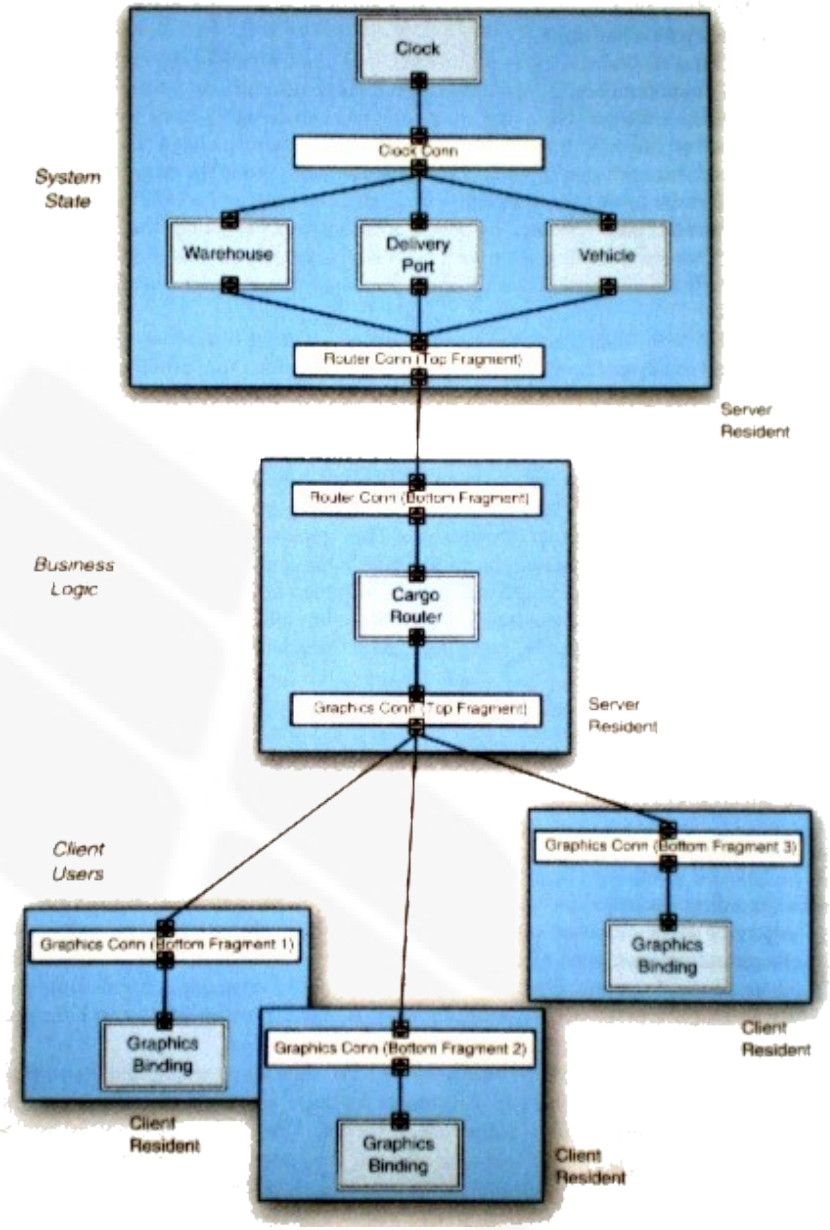
Perspectivas Arquiteturais

- Exemplo: perspectiva estrutural
  - Nenhuma informação sobre comportamento, interações etc



Perspectivas Arquiteturais

- Exemplo: perspectiva de implantação
  - Importante na avaliação da capacidade de satisfação dos requisitos
  - Ex: muitos componentes pesados em uma máquina com pouca memória e CPU modesta
  - Ex: transferência de altos volumes de dados em redes com baixa largura de banda



Arquitetura de Software

- Outras definições de arquitetura de *software*:

Arquitetura de Software = { ***elements*** (de processamento, dados ou conexão), ***form***, ***rationale*** (intenções, pressupostos, escolhas sutis, restrições externas, estilos e padrões adotados, etc) }

[Perry & Wolf - 1992]

Arquitetura de Software: é a organização fundamental do sistema, implementada por seus componentes, os relacionamentos entre eles e deles com o ambiente, e os princípios que governam seu projeto e evolução

[ANSI/IEEE Standard 1471-2000]

A Arquitetura de Software de um sistema já implantado é determinada pelos seus aspectos mais difíceis de serem modificados

[Chris Verhoef - 2005]

Componente

- Elementos de uma arquitetura geralmente implementam:
  - Processamento: funcionalidade ou comportamento
  - Estado: informação ou dados
  - Interação: inter-conexão, comunicação, coordenação e mediação
- Os componentes de *software* lidam com os dois primeiros problemas:

Componente de Software: é uma entidade arquitetural que: *1*) encapsula um subconjunto das funcionalidades e/ou dados do sistema; *2*) restringe o acesso a este subconjunto através de interfaces explicitamente definidas; e *3*) possui dependências - explicitamente definidas – em relação ao seu contexto de execução

Componente

- Um componente é um *locus* de computação e estado em um sistema [Shaw et al. - 1995]
- Pode ser simples como uma única operação ou complexo como um sistema inteiro
- É visto pelo usuário (humano ou outro *software*) somente através da sua interface pública
- São aplicações dos princípios de encapsulamento, abstração e modularidade

Componente

- O tratamento explícito do contexto de execução do qual o componente depende pode informar:
  - Interfaces requeridas pelo componente (*required interfaces*): serviços disponibilizados por outros componentes e dos quais o componente em questão depende para o seu correto funcionamento
  - Recursos necessários: arquivos de dados ou diretórios necessários ao componente
  - *Softwares* do sistema requeridos: ambientes de *run-time* plataformas de *middleware*, sistemas operacionais, protocolos de rede, *drivers* de dispositivos, etc
  - Configurações de *hardware* necessárias para executar o componente

Componente

- Geralmente são *application-specific*, mas não é sempre o caso (ex: servidores *web*, *front-ends*, *back-ends*, *toolkits* para GUI, componentes COTS, etc)
- Outra definição de componentes de *software*:

Componente de Software: unidade de composição formada somente de interfaces definidas de forma contratual e dependências explícitas de contexto

[Szyperski - 1997]

Conector

- Sistemas modernos são formados por um grande número de componentes complexos, distribuídos em múltiplos *hosts* (possivelmente móveis) e atualizados dinamicamente sem interrupção do serviço
- Nestes sistemas garantir uma interação apropriada pode ser mais importante e desafiador do que a implementação dos componentes

Conector de Software: elemento arquitetural responsável por efetivar e regular as interações entre componentes

Conector

- Em sistemas *desktop* convencionais os conectores são geralmente representados por simples chamadas de procedimento (*procedure call*) ou acesso a dados compartilhados (*Shared Data Access*)
- São constantemente não representados nas arquiteturas e se resumem a um meio de permitir a interação entre pares de componentes
- Entretanto, em sistemas complexos os conectores passam a ter identidades, papéis e artefatos de implementação únicos
- São elementos críticos, ricos e sub-apreciados

Conector

- O tipo mais simples e mais amplamente utilizado de conector é o *Procedure Call*, que permitem troca síncrona (bloqueante) de dados e controle entre pares de componentes
- Outro tipo comum de conector é o *Shared Data Access*, representado por alguma forma de variável não-local ou segmentos de memória compartilhada
  - Permite que múltiplos componentes se comuniquem assincronamente através da escrita e leitura de dados na área compartilhada

Conector

- Conectores do tipo *Distributor* encapsulam APIs para comunicação em rede, em sistemas distribuídos. Geralmente encapsulam outros conectores mais simples, como *Procedure Call*
- Conectores do tipo *Adaptor* são utilizados para integrar e permitir a interação entre componentes com interfaces e comportamentos incompatíveis
- Conectores são geralmente *application-independent*
  - Representam comportamentos já amplamente conhecidos, tais como: *"publish/subscribe", "notificação assíncrona de eventos"* e *"remote procedure call"*

Configuração

Configuração Arquitetural: conjunto de associações específicas entre os componentes e os conectores de uma arquitetura de *software*

- Geralmente representada por um grafo onde nós são componentes e conectores e arestas ligações
- Indica uma **possível** comunicação entre componentes, mas não garante a real habilidade deles se comunicarem
- As ligações devem ser entre interfaces compatíveis. Caso contrário tem-se um *architectural mismatch*

Estilo Arquitetural

- Experiências prévias podem evidenciar certas decisões arquiteturais que regularmente levam a projetos melhores
- Exemplo: as decisões abaixo têm garantido a disponibilização eficiente de serviços em sistemas distribuídos multi-usuário:
  - Separe fisicamente os componentes utilizados para requisitar daqueles utilizados para prover o serviço
  - Mantenha os provedores de serviço desconhecedores da identidade do requisitante
  - Requisitantes não devem ter contato uns com os outros
  - Permita que múltiplos provedores possam dinamicamente integrar o sistema

Estilo Arquitetural

- As decisões anteriores se aplicam a qualquer sistema neste contexto de disponibilização de serviços distribuídos
- Não são definidos detalhes acerca de componentes utilizados, suas interfaces e seus mecanismos de interação
- O arquiteto deve detalhar estas decisões e adaptá-las para o contexto específico de uma aplicação em particular
- Embora em alto nível de abstração, tais decisões definem o *rationale* subjacente à arquitetura

Estilo Arquitetural

Estilo Arquitetural: coleção identificada de decisões arquiteturais de projeto que: *1*) são aplicáveis a um determinado contexto de desenvolvimento; *2*) restringe as decisões arquiteturais específicas de um sistema em particular dentro deste contexto; e *3*) induz qualidades benéficas nos sistemas resultantes

- O exemplo anterior é uma descrição informal e parcial do estilo arquitetural *Client-Server*
- Outros exemplos: REST, *Pipe-and-Filter*

Padrão Arquitetural

- Estilos arquiteturais definem decisões gerais de projeto que impõem restrições e que podem precisar ser detalhadas em decisões mais específicas para o sistema em questão
- Em contraste, os padrões arquiteturais definem decisões de projeto consideradas eficientes para certas classes de sistemas e que podem ser **configurados** com os componentes e conectores do sistema em questão

Padrão Arquitetural: coleção identificada de decisões arquiteturais de projeto que são aplicáveis a um problema recorrente de desenvolvimento e parametrizadas de modo a serem aplicadas em qualquer contexto de desenvolvimento de *software* no qual o problema aparece

Padrão Arquitetural

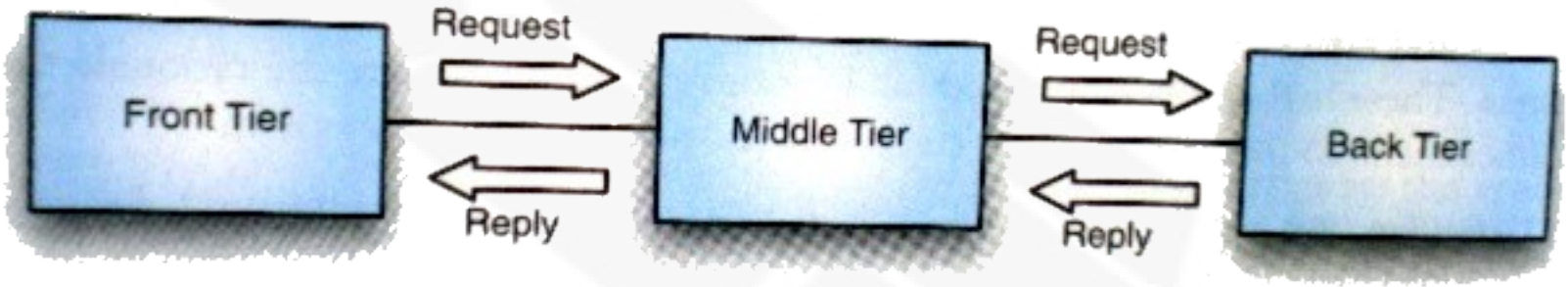
- As definições de estilo e padrão arquitetural são parecidas, nem sempre poderemos fazer a distinção com clareza
- Porém, estilos e padrões diferem em alguns aspectos
  - Escopo: estilos se aplicam a um **contexto** de desenvolvimento (ex: sistemas altamente distribuídos, sistemas *GUI-intensive*), enquanto padrões se aplicam a um **problema** de projeto específico (ex: o estado do sistema deve ser apresentado de múltiplas formas, a camada de negócio deve ser separada da camada de dados)
    - Um problema é mais concreto que um contexto
    - Estilos são *estratégicos* enquanto padrões são *táticos*

Padrão Arquitetural

- Porém, estilos e padrões diferem em alguns aspectos
  - Abstração: um estilo ajuda a restringir as decisões arquiteturais do sistema. Requer intervenção humana para relacionar as diretrizes ditadas pelo estilo com os problemas de projeto do sistema em questão
    - Por si só, são muito abstratos para já representar um projeto concreto do sistema
    - Padrões, por sua vez, são fragmentos arquiteturais parametrizados e podem ser considerados como peças concretas do projeto
  - Relacionamento: o mesmo padrão pode ser aplicado em sistemas que seguem diferentes estilos
    - Um sistema que segue um determinado estilo arquitetural pode envolver o uso de vários padrões arquiteturais

Padrão Arquitetural

- Exemplo de padrão arquitetural:
  - Sistema em três camadas



- *Front end (tier)*: contém as funcionalidades necessárias para acessar o serviço (GUI + *cache* + processamento mínimo)
- Camada de aplicação (*middle*): processa requisições do *front-end* e acessa e processa os dados do *back-end*
- *Back end (tier)*: contém as funcionalidades para armazenamento e acesso a dados
- Interações seguem o paradigma *request-reply*, porém nada além disso é prescrito (síncrono ? r*equest-triggered* ? *single-request-single-reply* ?)

Padrão Arquitetural

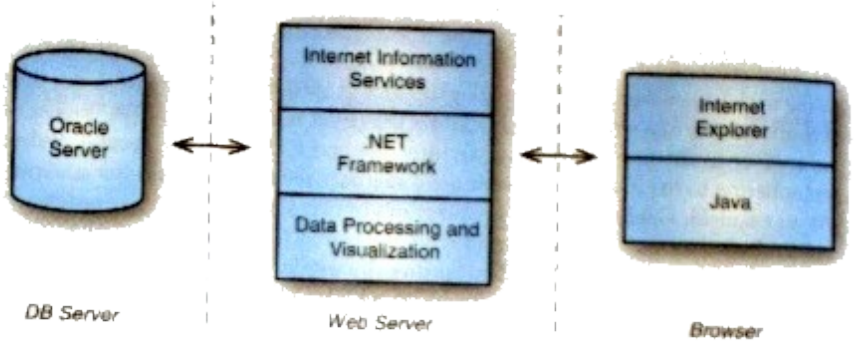
- Exemplo de padrão arquitetural:
  - Sistema em três camadas: parâmetros
    - Quais facilidades para interface de usuário, processamento, armazenamento e acesso a dados – específicos da aplicação – são necessários ?
    - Como elas devem ser organizadas dentro de cada camada ?
    - Quais mecanismos devem ser utilizados para permitir a interação entre as camadas ?

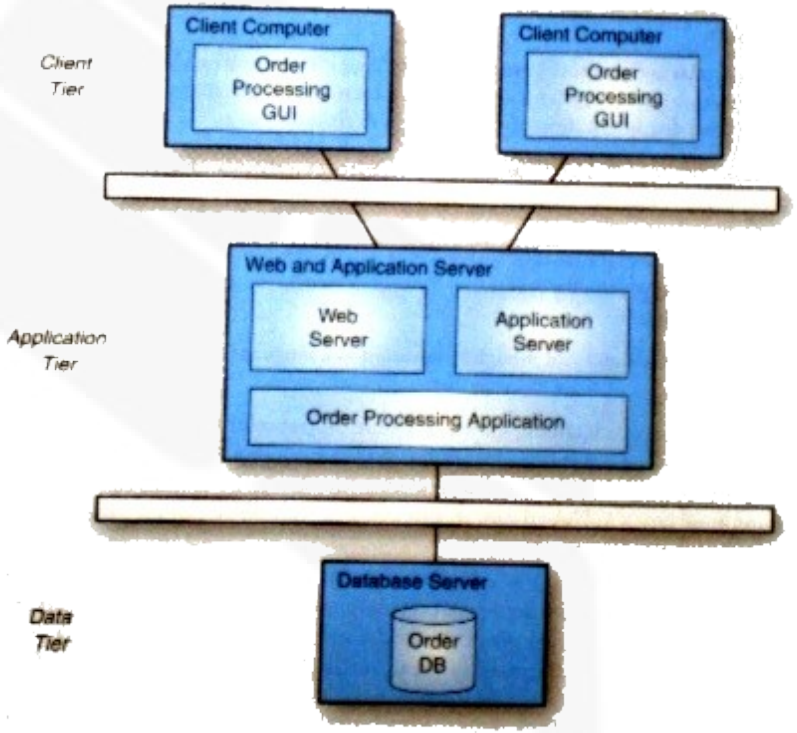
Padrões X Estilos

- Estilos demandam mais atenção do arquiteto e disponibilizam um suporte não tão direto quanto os padrões
- O padrão *Sistema em Três Camadas* pode ser visto como a sobreposição de duas arquiteturas que seguem o estilo *Client-Server*

Padrões X Estilos

- Dois sistemas em três camadas diferentes:
  - Quais traços arquiteturais são comuns ?
  - Quais são diferentes ?





Modelos

- A arquitetura de um *software* é capturada em um modelo arquitetural, utilizando alguma notação de modelagem

Modelo Arquitetural: artefato que captura algumas ou todas as decisões de projeto que compõem a arquitetura do sistema

Modelagem Arquitetural: atividade que reifica e documenta estas decisões arquiteturais de projeto

  - Um sistema pode ter diversos modelos associados
  - Os modelos podem variar em relação à quantidade de detalhes capturados, perspectiva utilizada, tipo de notação, etc

Modelos

Notação de Modelagem Arquitetural: é uma linguagem ou um meio de captura das decisões arquiteturais de projeto

- Frequentemente chamadas de ADLs (*Architectural Description Languages*)
  - ADLs podem ser textuais ou gráficas, informais (diagramas em *slides*), semi-formais (UML), formais, de domínio específico ou propósito geral, proprietárias ou padronizadas, etc
- Modelos arquiteturais são artefatos críticos e servem como base para a realização das tarefas subsequentes

Processos

- Arquitetura não é uma fase e sim um elemento integrado a todas as fases e por elas influenciado
- Serve para integrar as diferentes atividades:
  - Projeto arquitetural
  - Modelagem arquitetural e Visualização arquitetural
  - Análise de sistemas orientada a arquiteturas
  - Implementação de sistemas orientada a arquiteturas
  - Implantação, re-implantação em *run-time* e mobilidade de sistemas orientados a arquiteturas
  - Projeto de propriedades não-funcionais baseado em arquiteturas
  - Adaptação arquitetural

Processos

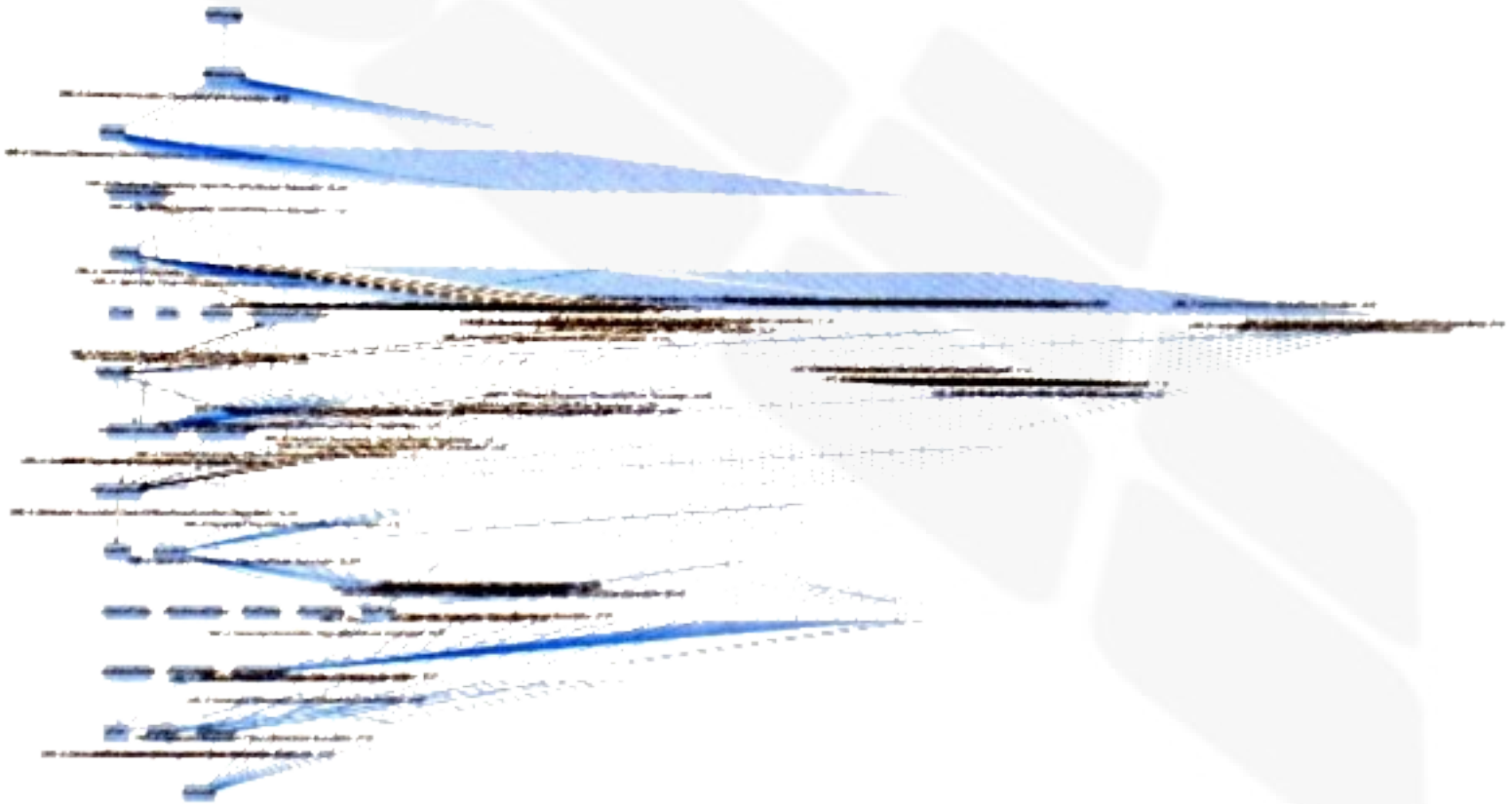
- Recuperação Arquitetural:
  - Com degradações constantes chegará o momento onde mudanças adicionais no sistema se tornam inviáveis
  - Os efeitos das mudanças também se tornam imprevisíveis visto que a arquitetura prescritiva está tão desatualizada que se torna inútil, podendo até atrapalhar o processo
  - Realiza-se então a recuperação arquitetural:

Recuperação Arquitetural: processo de determinação da arquitetura de um sistema a partir dos seus artefatos de implementação

  - Artefatos = código-fonte, executáveis, *byte-codes*, etc

Processos

- Recuperação Arquitetural:
  - Diagrama gerado diretamente do código-fonte:



Stakeholders

- Pessoas envolvidas e interessadas no projeto:
  - Arquiteto de *software*: define, modela, avalia e evolui a arquitetura do sistema. Mantém a integridade conceitual do sistema. É um *stakeholder* crítico
  - Desenvolvedores: consumidores primários dos produtos do arquiteto. Implementam as decisões de projeto presentes na arquitetura gerando a implementação do sistema
  - Gerente de *software*: supervisiona o projeto e apoia o arquiteto. Se necessário, exerce sua autoridade em nome do arquiteto
  - Clientes: desejam um sistema de alta qualidade, satisfazendo os requisitos no prazo e dentro dos custos estimados

INF016 – Arquitetura de Software

03 – Conceitos Básicos

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas