Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software

Sandro S. Andrade
sandroandrade@ifba.edu.br

Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software Introdução

Sandro S. Andrade
sandroandrade@ifba.edu.br

Introdução

- A área de Arquitetura de Software estuda como sistemas de software são projetados e construídos
- Arquitetura de Software:
- Conjunto formado pelas principais decisões de projeto tomadas durante seu desenvolvimento e qualquer evolução subsequente
- Desenvolvimento de software centrado em arquiteturas
- Qualidade de projeto → Qualidade de Software
- Famílias de produtos de software

Software e Construção Civil

- Analogia forte e de fácil compreensão
- As fases são similares (requisitos, projeto, etc)
- Outras similaridades:
- Projeto arquitetural com foco nas necessidades dos usuários
- Permite especialização de trabalho
- Planos e progressos podem ser avaliados em pontos intermediários
- Mas não é só isso ...

Software e Construção Civil

1) Toda construção tem uma arquitetura, separada, porém relacionada, à estrutura física
   - Esta arquitetura pode ser descrita, discutida e comparada com as de outras construções
   - A arquitetura antecipadamente projetada pode ser comparada com a arquitetura resultante do processo de construção
   - De forma similar, a arquitetura de um software existe de forma independente, porém relacionada, ao código-fonte que a implementa

Software e Construção Civil

2) Propriedades das estruturas são induzidas pelo projeto das suas arquiteturas:

- Castelo medieval: paredes altas e espessas e janelas estreitas, se existentes. Induz propriedades defensivas
- Propriedades de um software, como resiliência a tipos particulares de ataques, são determinadas pelo projeto de suas arquiteturas

Software e Construção Civil

- Objetivos de uma boa arquitetura:
  - Força: fundações para um assoalho sólido, escolha apropriada de materiais sem economia
  - Utilidade: distribuição sensata das partes, com seus propósitos devidamente atendidos e situação apropriada
  - Beleza: aparência agradável, boa percepção do "todo" e dimensões proporcionais entre as partes

Software e Construção Civil

3) Reconhecimento do papel distinto e característico do arquiteto – pessoa que cria a arquitetura

- Exige ampla formação:
  - Aspectos de engenharia
  - Senso apurado de estética
  - Conhecer o modo como as pessoas trabalham, comem, brincam e moram ajuda a projetar construções satisfatórias e que funcionam bem ao longo das estações e dos anos
  - Habilidades simples de programação não são suficientes para a criação de sistemas complexos que efetivamente funcionam

Software e Construção Civil

4) O processo não é tão importante quanto a arquitetura

- Isso não quer dizer que o processo não é importante, somente que ele não é garantia de sucesso
- O processo existe para servir um fim – o projeto e a qualidade da infra-estrutura – não para ser um fim em si próprio

Software e Construção Civil

5) A arquitetura (de software) amadureceu, ao longo dos anos, como uma disciplina

- Uma base de conhecimentos está disponível, capturando as experiências e lições de projeto prévios
- Foco no reuso de conhecimento, de projeto de sub-sistemas e de ferramentas
- Benefício de uso de materiais, partes e tamanhos padronizados

Software e Construção Civil

- Estilos Arquiteturais:
  - Vila Romana
  - Catedral Gótica
  - Estilo fazenda
  - Chalé Suiço, Arranha-céu
- Tentam encontrar um conjunto comum de requisitos e acomodar as restrições de topologia local, clima e materiais, ferramentas e mão-de-obra disponíveis
- Um estilo coloca restrições ao desenvolvimento, o que leva a qualidades particulares desejáveis

Software e Construção Civil

- Limitações da analogia:
  - Conhecemos muito sobre prédios e não tanto sobre software
  - Natureza essencial dos materiais totalmente diferente
  - O software é mais "maleável" do que os materiais físicos de construção
  - A indústria da construção civil é mais consolidada
  - O fase de implantação não existe na construção civil
  - Caráter extremamente dinâmico do software

Software e Construção Civil

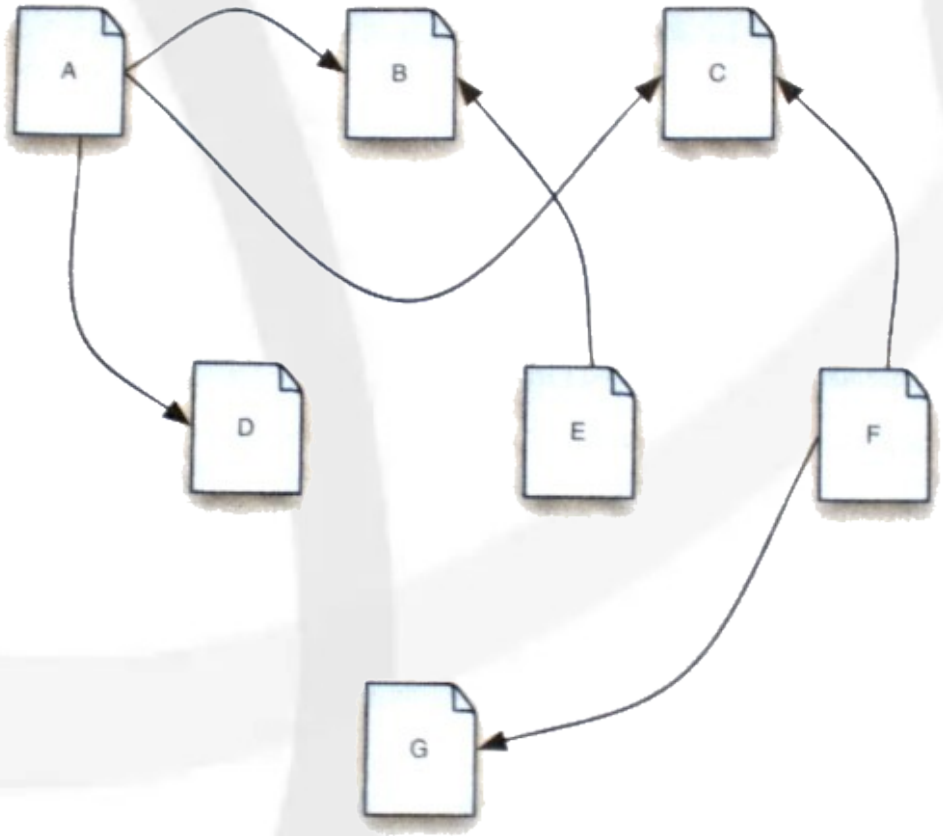
- Resumindo:
  - A arquitetura do software deve ser o centro do projeto e desenvolvimento de sistemas, mais importante que o processo, análise e até mesmo programação
  - Ao dar proeminência à arquitetura obtém-se: controle intelectual, integridade conceitual, base adequada e efetiva para reuso, comunicação efetiva no projeto e gerenciamento de um conjunto de sistemas variantes, porém relacionados
  - O foco na arquitetura deve estar presente em todas as fases do projeto

Exemplos

- Exemplo 1: Arquitetura da web
  - O que é a web ? Como ela é construída ? Como você projetaria um software para um site de comércio eletrônico ?
  - A arquitetura do sistema fornece o vocabulário e os meios para responder as questões acima, em particular o estilo arquitetural adotado para a web

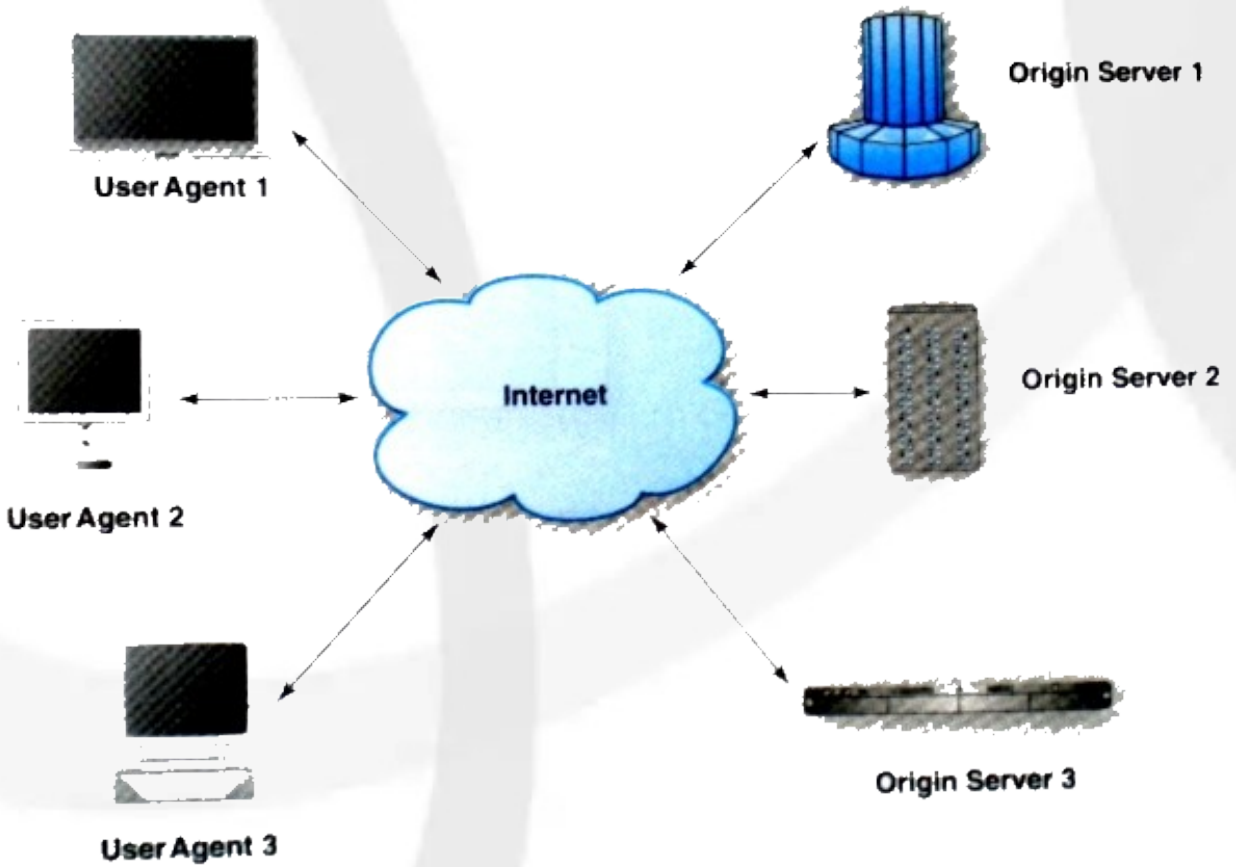
Exemplos

- Visão do usuário: conjunto dinâmico de relacionamentos entre coleções de informação



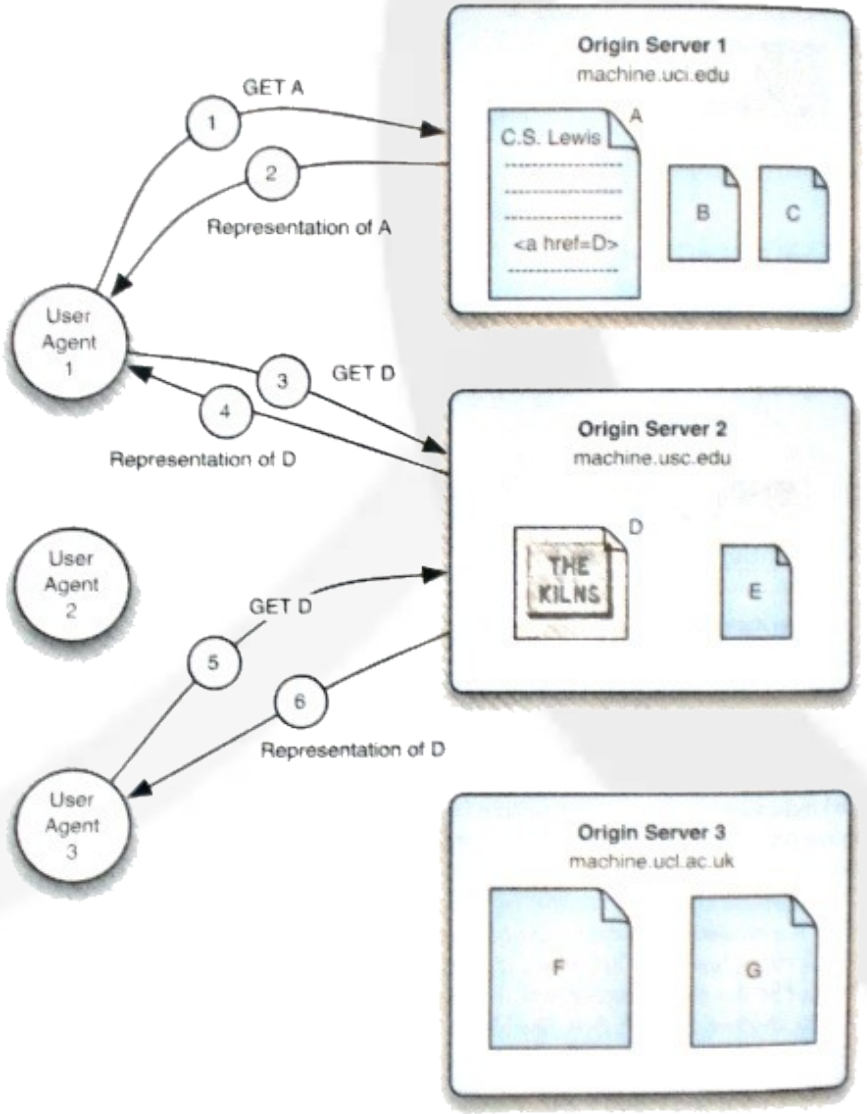
Exemplos

- Visão de rede: coleção de máquinas independentemente apropriadas e operadas, que se comunicam via rede



Exemplos

- Visão do desenvolvedor: coleção de programas independentemente desenvolvidos que se comunicam através dos padrões HTTP, URI, MIME e HTML



Exemplos

- Estas visões não explicam como a web funciona
- Uma estratégia melhor é apresentar um conjunto de definições e restrições que caracterizam a web:
  - Coleção de resources, identificados unicamente por uma URL
  - Cada resource denota uma informação, como um documento, imagem, serviço, coleção de outros resources, etc
  - URL's podem ser utilizadas para determinar a identidade da máquina que contém o resource
  - Toda comunicação é iniciada pelos clientes (user agents), realizando requisições aos servidores

Exemplos

- Uma estratégia melhor é apresentar um conjunto de definições e restrições que caracterizam a web:
  - Resources podem ser manipulados através de suas representações. O HTML é a linguagem de representação mais comum da web
  - Toda comunicação entre clientes e servidores é realizada através de um protocolo extremamente simples (HTTP), com poucas primitivas, tais como GET e POST
  - Toda comunicação entre clientes e servidores é context-free, ou seja, o servidor responde à requisição baseando-se somente na informação presente na própria requisição. Nenhum histórico de operações é mantido

Exemplos

- Exemplo 2: Shell Script

ls invoices | grep -e August | sort

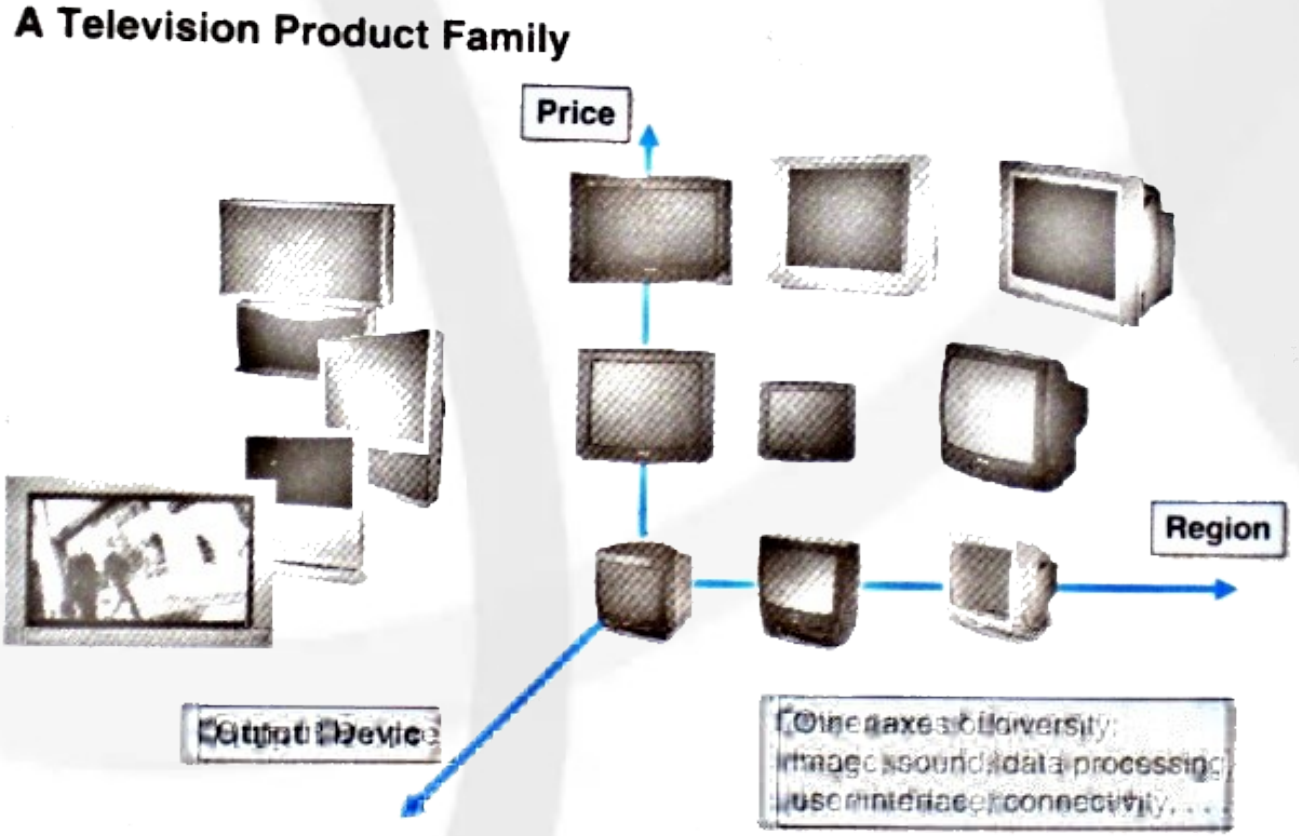
  - Um filtro é um programa que recebe um fluxo de caracteres como entrada e produz um fluxo de caracteres como saída. Filtros podem ser parametrizados
  - Um pipe é uma forma de conectar dois filtros, onde a saída do primeiro filtro é conectada à entrada do segundo
  - Conhecendo os filtros e pipes utilizados pode-se facilmente compreender o programa e criar outros
  - O conjunto particular de regras aqui aplicado define um estilo arquitetural conhecido como Pipe-and-Filter
  - Pode ser utilizado em qualquer sistema

Exemplos

- Exemplo 3: Linhas de Produto
  - Famílias de produto são conjuntos de programas independentes que possuem um alto potencial de compartilhamento de estrutura e componentes constituintes
  - Ex: HD TV 35" com DVD player com sinal ATSC, HD TV 35" sem DVD player com sinal ATSC, HD TV 35" com DVD player com sinal DVB-T
  - Reutilizar estruturas, comportamentos e implementações simplifica o desenvolvimento, reduz prazos e custos e melhora a confiabilidade geral do sistema
  - Arquiteturas de software são abstrações essenciais para o gerenciamento de variações e de pontos em comum

Exemplos

- Linha de produtos da Philips
  - Metodologia arquitetural: Koala

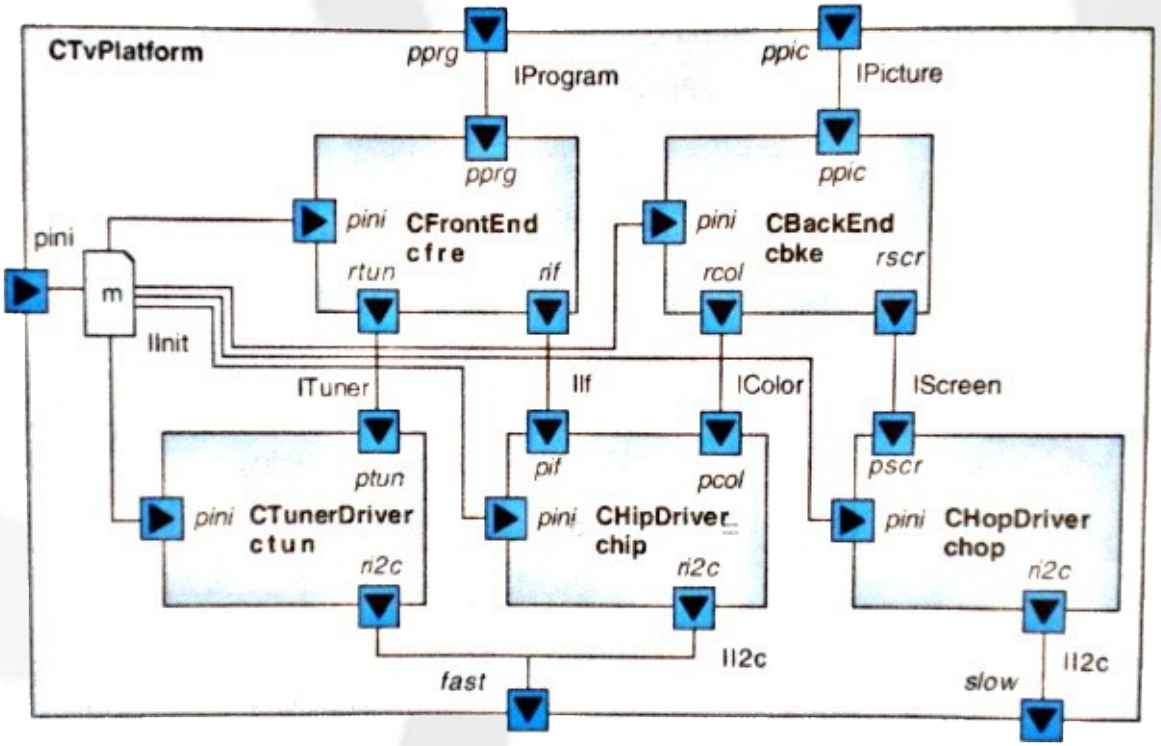


Exemplos

- Koala:
  - Modela e implementa o software como uma coleção de componentes que interagem entre si
  - Cada componente exporta um conjunto de serviços através de um conjunto de provided interfaces
  - Cada componente explicitamente define suas dependências com o ambiente (hardware ou software) através de um conjunto de required interfaces

Exemplos

- Arquitetura exemplo de uma plataforma de TV da Philips:
  - Compatibilidade de interfaces
  - Composite

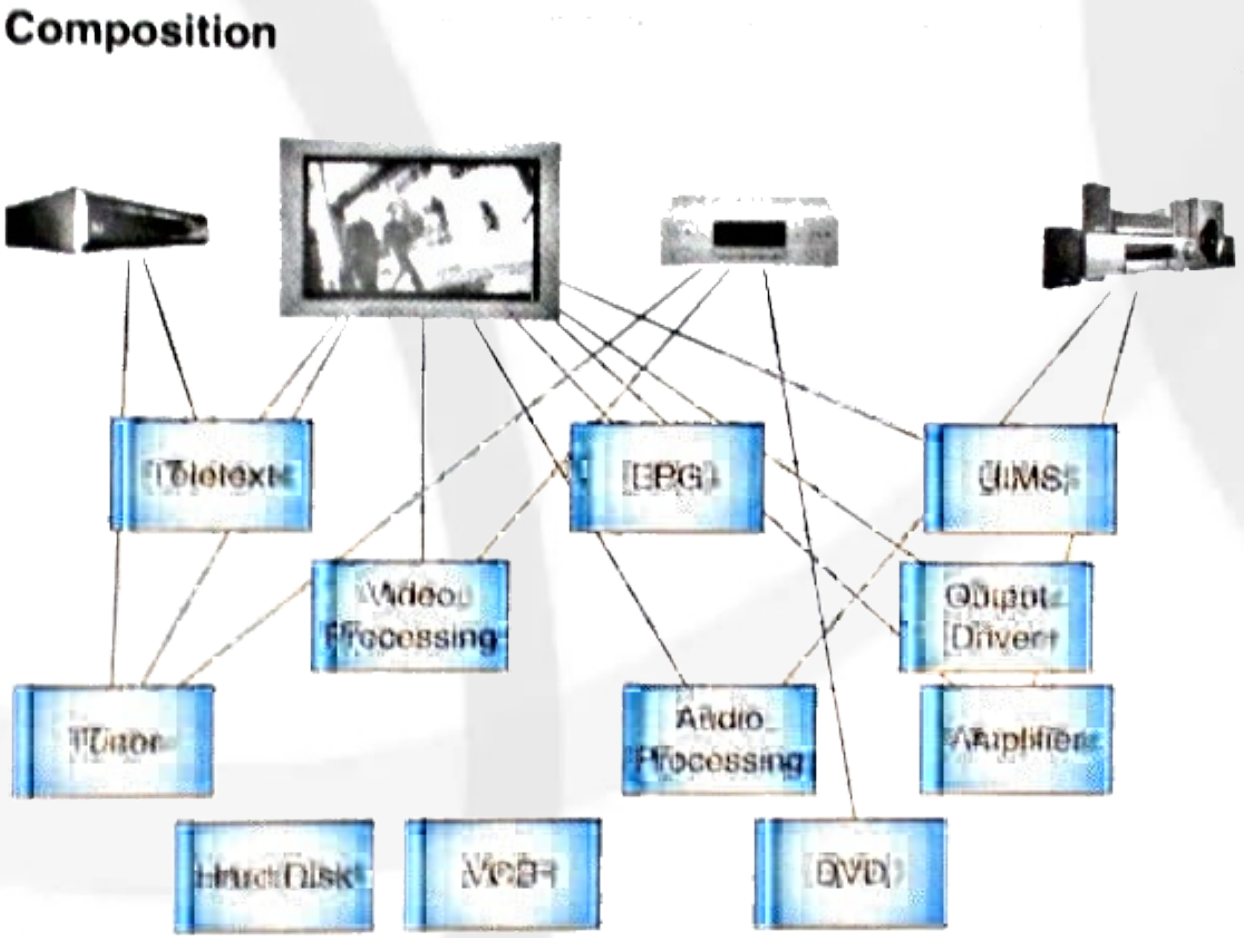


Exemplos

- Koala: mecanismos para gerenciamento da variabilidade:
  - Diversity interfaces: mecanismo para parametrizar um componente. Permite que um componente importe propriedades específicas da configuração a partir de elementos do Koala que implementam esta interface. São externos ao componente
  - Switches: elemento de conexão que permite que um componente interaja com apenas um dentre um conjunto de componentes, dependendo do valor de um parâmetro obtido em run-time
  - Optional interfaces: provê ou requer funcionalidades presentes em apenas alguns produtos da família

Exemplos

- Koala: população de produtos



Exemplos

- Famílias de produtos demandam mudanças nos processos e práticas organizacionais
- A abordagem padrão de desenvolvimento não suporta linhas de produto de forma eficiente
- O Koala é a manifestação concreta da experiência corporativa, conhecimentos e vantagens competitivas da empresa
- Arquiteturas de software evitam a dependência de uma estrutura social (permanência de funcionários, etc) e suportam o alcance de níveis maiores de produtividade

Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software Reorientação de Engenharia

Sandro S. Andrade
sandroandrade@ifba.edu.br

Arquiteturas em Contexto

- Como a arquitetura de software se relaciona com os conceitos de engenharia de software tradicionalmente aplicados ?
- Tais conceitos devem ser re-orientados, pois a arquitetura do software passa a ocupar papel fundamental
- Conhecimentos fundamentais:
  - Toda aplicação tem uma arquitetura
  - Toda aplicação tem ao menos um arquiteto
  - Arquitetura não é uma fase do desenvolvimento

Arquiteturas em Contexto

- Questões consequentes:
  - De onde surge a arquitetura de uma aplicação ?
  - Como uma arquitetura de software pode ser caracterizada ?
  - Quais são as suas propriedades ?
  - É uma arquitetura boa ou ruim ?
  - Suas deficiências podem ser facilmente corrigidas ?
  - Os arquitetos estão sempre conscientes das decisões fundamentais de projeto que tomam ?
  - Essas decisões de projeto podem ser articuladas com outras ?

Arquiteturas em Contexto

- Questões consequentes:
  - Os arquitetos conseguem manter a integridade conceitual do projeto ao longo do tempo ?
  - Alternativas foram consideradas nos diversos momentos de decisão ?
- A arquitetura do software não é produto de uma fase específica do processo, realizada após a análise de requisitos e antes do projeto detalhado
- A criação e manutenção da arquitetura estão presentes em todo o processo, embora tenham destaque especial em uma fase particular

Análise de Requisitos

- Considerações sobre a arquitetura começam no início do projeto
- Noções de estrutura, projeto e solução são completamente apropriadas durante a fase de análise de requisitos
- Visão tradicional: a análise e especificação de requisitos deve permanecer isolada de qualquer consideração sobre qual projeto irá satisfazer os requisitos
  - "A parte central deste artigo esboça uma abordagem para análise de requisitos que evita a atração magnética da orientação à solução" [Jackson 2000]

Análise de Requisitos

- Abordagem de George Polya em "How to solve it" - 1957:
  - Primeiro compreenda o problema
  - Depois encontre uma conexão entre os dados e o desconhecido. Em algum momento você vai encontrar um plano para a solução
  - Execute o plano
  - Examine a solução obtida
- Ambas abordagens defendem a completa exploração e compreensão dos requisitos antes de propor uma solução

Análise de Requisitos

- Exemplo de uso desta abordagem: máquinas de lavar
  - Máquinas de lavar não batem roupas em rochas à beira de um rio
  - O foco nos requisitos, sem qualquer atração pela "orientação à solução", permitiu a obtenção de soluções novas e criativas: tambores rotativos com agitadores
- Na prática, entretanto, a análise de requisitos é realizada de modo rápido e superficial:
  - Restrições de orçamento e prazos
  - Processos de desenvolvimento inferiores
  - Falta de confiança nos engenheiros responsáveis

Análise de Requisitos

- Os motivos, entretanto, envolvem limitações humanas em relação a racionínio abstrato, economia e evocação
- Analogia com construção de casas:
  - Não raciocinamos sobre nossas necessidades independente de como elas serão satisfeitas
  - Pensamos em número de cômodos, estilo das janelas, fogão a gás ou elétrico
  - Não pensamos em termos de "uma forma de prover abrigo a um clima rigoroso", "uma forma de prover iluminação adequada" ou "uma forma de preparar comida aquecida"

Análise de Requisitos

- O mesmo acontece com software:
  - Sem referência a arquiteturas já existentes torna-se difícil avaliar a viabilidade, cronograma e custo do projeto
  - Conhecer as interfaces de usuário, hardware e tipos de serviços disponíveis ajuda a chegar em requisitos baseados numa compreensão razoável da viabilidade
  - As falhas impulsionam a engenharia e são a base para inovação: observação + detecção das limitações
  - Exemplo: criação do zipper – sucessor de uma longa sequência de invenções
  - "Como muitos outros produtos, o zipper não surgiu diretamente das funcionalidades mas de correções sucessivas de falhas" [Petroski 1992]

Análise de Requisitos

- Observações fundamentais:
  - Projetos e arquiteturas já existentes definem um vocabulário para discutir as possibilidades
  - Nossa compreensão sobre o que funciona hoje e como ele funciona afeta nossos desejos – foco na solução
  - Experiências prévias com sistemas nos ajuda a avaliar a viabilidade e definir custos e prazos
  - Requisitos = articulação de melhorias necessárias à arquitetura vigente
  - Isso não significa limitar inovação, as máquinas de lavar foram progressivamente aperfeiçoadas

Análise de Requisitos

- Observações fundamentais:
  - Não se limitar, entretanto, aos projetos atuais. Diferentes mecanismos devem ser usados para subir em uma casa, arranha-céu ou até a lua
  - Quando não existem antecessores físicos analogias ou antecessores conceituais podem ajudar
  - Desenvolvimento greenfield também utiliza antecessores para enquadrar os requisitos e soluções inéditos
  - Nem todas as arquiteturas, entretanto, são boas fontes de inspiração

Projeto

- Fase onde maior atenção é dada à definição das principais decisões de projeto (arquitetura)
- O desenvolvimento da arquitetura, entretanto, não é exclusivo desta fase
- Projeto é um aspecto de todas as outras atividades de desenvolvimento
- Decisões arquiteturais refletem vários aspectos do sistema, requerendo um rico "repertório" de técnicas de projeto

Projeto

- Modelo tradicional (em cascata):
  - Objetivo: definir um projeto que possa ser repassado para os programadores
  - Se algum requisito é considerado inviável, retorna-se à fase de análise de requisitos (sem entretanto retomar aspectos da solução)
  - Se, na implementação, alguma parte do projeto é considerada inviável, retorna-se à fase de projeto

Projeto

- Modelo centrado em arquiteturas:
  - Reduz-se ou elimina-se as fronteiras entre as fases, geralmente artificiais e improdutivas
  - Análise de requisitos já relacionada a aspectos de arquitetura e projeto
  - Análise, projeto e implementação acontecem de uma forma mais integrada e enriquecida
- O arquitetura lida com uma ampla faixa de questões:
  - Interesses dos stakeholders, estilo e estrutura utilizados, tipos de conectores de elementos, estrutura primárias de classes e pacotes, aspectos de distribuição, descentralização, implantação e segurança, etc

Projeto

- Técnicas de projeto de sistemas:
  - Projeto Orientado a Objetos:
    - Identificação de objetos que encapsulam um estado e funções que acessam e manipulam este estado
    - Não é uma abordagem completa nem é eficaz em todas as situações. Não é ineficaz, entretanto
  - Limitações:
    - Não é uma abordagem completa de projeto. Não aborda questões de implantação, segurança, confiança, etc
    - Não possui mecanismos para transferir, para novas arquiteturas, conhecimento de domínio e soluções presentes em arquiteturas anteriores
    - Exige que todos os conceitos e entidades sejam objetos. Não há suporte explícito para algo que não seja uma classe

Projeto

- Técnicas de projeto de sistemas:
  - Projeto Orientado a Objetos:
    - Limitações:
      - Disponibiliza somente um tipo de encapsulamento (objeto), uma noção de interface, um único tipo de conector explícito (procedure call). Não suporta a noção de required interfaces
      - Fortemente ligada a interesses e decisões das linguagens de programação, que podem começar a ditar quais decisões são importantes
      - Assume um espaço de endereçamento compartilhado e suporte adequado ao gerenciamento de heap e stack
      - Assume a existência de uma única thread de controle
      - Aspectos de concorrência, distribuição e descentralização não são considerados
    - A UML ajudou a discutir um projeto orientado a objetos sem depender da linguagem de programação

Projeto

- Técnicas de projeto de sistemas:
  - Domain-Specific Software Architectures (DSSAs)
    - Apropriada quando experiências e arquiteturas anteriores influenciam potencialmente os novos projetos
    - Geralmente a experiência traz a melhor abordagem e melhor solução para o domínio em questão
    - Novas arquiteturas serão variações das anteriores
    - Abre-se espaço para focar em variações originais e criativas
    - Reutiliza-se partes da arquitetura e da implementação
    - Exige bom suporte técnico: arquiteturas anteriores devem ser capturadas e refinadas para reuso, pontos de variação devem ser identificados e isolados, interfaces nos pontos de variação devem ser explícitas, etc

Implementação

- Objetivo: criar um código-fonte que seja fiel à arquitetura e que implemente de forma completa os requisitos
- A abordagem centrada em arquiteturas dá ênfase a algumas abordagens à implementação:
  - A implementação pode estender ou modificar a arquitetura
  - A arquitetura só estará completa após a implementação
  - Deve-se manter as decisões que constituem a arquitetura consistentes com o código-fonte produzido
  - Estimula a utilização de técnicas generativas e fortemente baseadas em reutilização (ex: uso de frameworks)

Implementação

- Implementação fiel:
  - Todos os elementos estruturais da arquitetura estão implementados no código-fonte
  - O código-fonte não deve utilizar elementos computacionais que não estão presentes na arquitetura
  - O código-fonte não deve conter conexões (entre elementos da arquitetura) que não estão descritas na arquitetura

Implementação

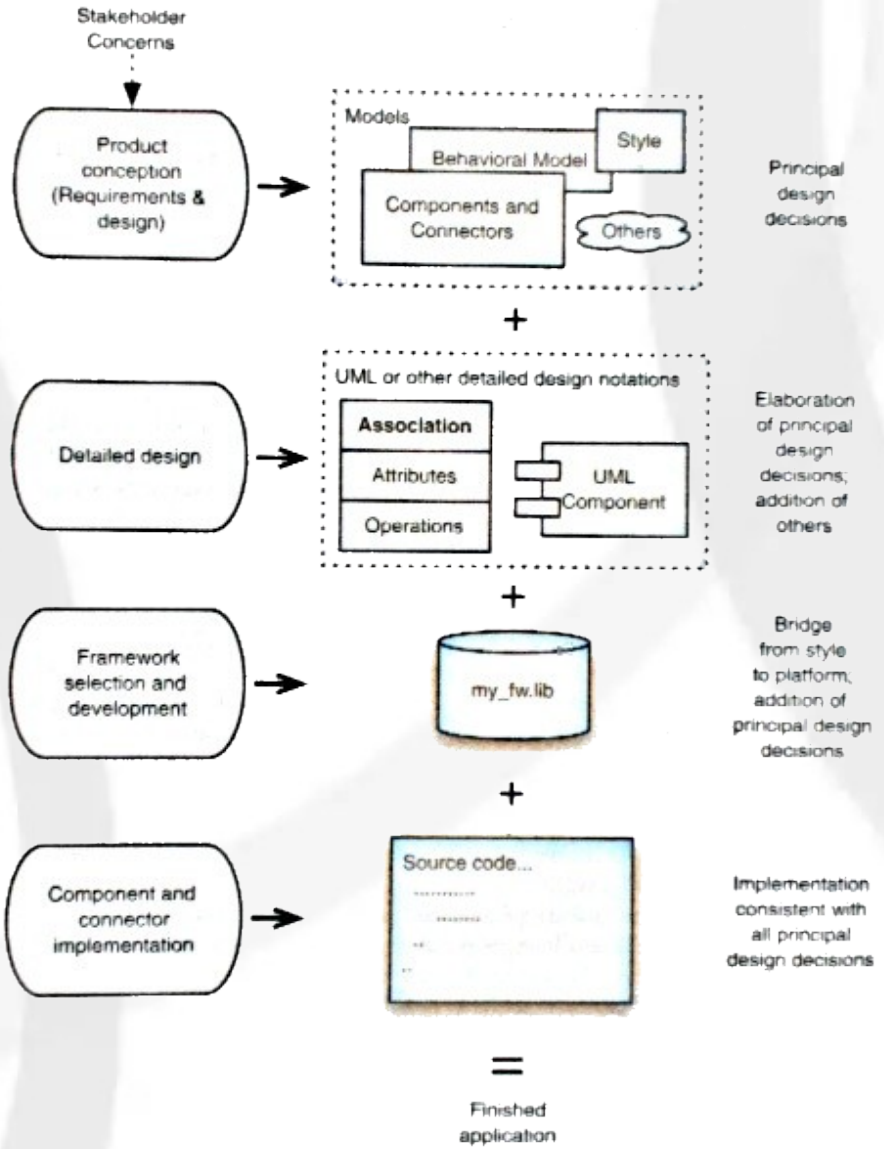
- Na prática essa definição não é totalmente adequada:
  - Uso de bibliotecas reduzem o custo e aumentam a qualidade, porém contêm funções e interfaces que não estão presentes na arquitetura
  - Se uma biblioteca barata e de qualidade implementa 98% da funcionalidade desejada e se as consequências da ausência dos outros 2% forem aceitáveis:
    - Decide-se pelo uso da biblioteca
    - Realiza-se a revisão das especificações e da arquitetura
  - O ponto crítico é sempre manter a arquitetura e a implementação em estados consistentes

Implementação

- Estratégias de implementação (em prioridade):
  - Técnicas generativas: a implementação é gerada automaticamente e possui alta qualidade
    - Geralmente aplicada em domínios muito específicos
  - Técnicas baseadas em reuso: demandam menor tempo e produzem código de maior qualidade do que construir o sistema do zero
    - Architecture Implementation Framework = ponte entre um estilo arquitetural específico e um conjunto de tecnologias de implementação. Traz garantias
    - Soluções de middleware, COTS, open-source
  - Desenvolvimento manual completo: custos e prazos maiores. Maior necessidade de garantia da qualidade

Implementação

- Architecture Implementation Framework:



Implementação

- Se a implementação difere da arquitetura projetada, esta arquitetura não caracteriza a aplicação
- O sistema tem uma arquitetura, porém latente, em contraste àquela documentada
- Falhas em reconhecer esta diferença:
  - Rouba a habilidade de raciocinar, no futuro, sobre a arquitetura implementada da aplicação
  - Engana os stakeholders em relação ao que eles "acreditam que têm" e o que eles "realmente têm"
  - Torna qualquer estratégia de desenvolvimento ou evolução, baseada na arquitetura, imprecisa e fadada ao fracasso

Análise e Teste

- Atividades realizadas para garantir a qualidade de um artefato
- Na abordagem tradicional o código-fonte é examinado em termos de corretude funcional e eventualmente desempenho
- Entretanto, análise de uma determinada propriedade pode ser realizada assim que o artefato existir, seja ele o que for
- Porque só o código-fonte é testado ?
- Porque o teste é realizado somente em relação aos requisitos funcionais da aplicação ?

Análise e Teste

- Resposta: devido à ausência de qualquer representação suficientemente rigorosa da aplicação que não seja o código-fonte
- Arquiteturas permitem uma análise antecipada e melhorada do código-fonte
- Abre-se caminho para a análise de propriedades não-funcionais
- Quais os benefícios que as arquiteturas de software trazem à fase de análise e teste ?

Análise e Teste

1) O modelo arquitetural pode ser avaliado em relação à sua consistência interna e corretude:
    - Verificações sintáticas do modelo podem identificar, por exemplo, conexões entre componentes não compatíveis (interface mismatch), especificação incompleta de propriedades e padrões de comunicação indesejados
    - Análise de fluxo de dados pode ser aplicada para determinar incompatibilidades de definição/uso e para detectar falhas de segurança
    - Técnicas de model-checking podem analisar problemas de deadlock
    - Técnicas de simulação podem realizar formas simples de análise dinâmica

Análise e Teste

2) O modelo arquitetural pode ser avaliado em relação à sua consistência com os requisitos:
   - Independente do processo utilizado o modelo arquitetural deve ser consistente com os requisitos
   - Esta verificação talvez precise ser feita de forma manual, caso os requisitos estejam descritos em linguagem natural
   - A verificação é essencial

Análise e Teste

3) O modelo arquitetural pode ser utilizado para determinar e suportar estratégias de análise e teste aplicadas ao código-fonte:
   - A arquitetura provê o projeto do código-fonte, consistência entre eles é essencial
   - A arquitetura serve como uma fonte de informação para governar testes, baseados na especificação, que atuam em todos os níveis: unidade, sub-sistema e sistema
     - O arquiteto pode priorizar análises e testes com base na arquitetura, focando os componentes e montagens mais críticos
     - Ex: componentes comuns em todos os membros de uma família de produtosn

Análise e Teste

3) O modelo arquitetural pode ser utilizado para determinar e suportar estratégias de análise e teste aplicadas ao código-fonte:

- A arquitetura serve como uma fonte de informação para governar testes, baseados na especificação, que atuam em todos os níveis: unidade, sub-sistema e sistema
- A arquitetura disponibiliza um meio para repassar, para novos projetos, resultados prévios de análise
- Ex: a extensão do teste de unidade de um componente é reduzido se ele está sendo aplicado em um mesmo contexto e condições de uso
- A arquitetura ajuda a desviar a atenção do analista para os conectores em uma implementação do sistema

Análise e Teste

4) O modelo arquitetural pode ser comparado com um modelo derivado a partir do código-fonte da aplicação:

- É uma forma de checar a sua solução
- Seja P um programa derivado da arquitetura A
- Um grupo diferente de engenheiros, sem acesso a A, desenvolve um modelo arquitetural A' a partir da análise de P
- Se tudo estiver correto A será consistente com A'
- Caso contrário, ou P não implementa A fielmente ou A' não reflete fielmente a arquitetura de P
- Em qualquer caso é necessário uma verificação

Evolução e Manutenção

- Evolução ou Manutenção de Software refere-se a todo tipo de atividade realizada após o lançamento (release) da aplicação
- A abordagem tradicional para evolução é ad-hoc, geralmente retorna-se à fase do processo relacionada à mudança
- O risco é a degradação da qualidade da aplicação:
  - Devido a mudanças realizadas em qualquer lugar, por qualquer meio que seja o mais rápido e mais fácil
  - Com o tempo, realizar mudanças sucessivas se torna extremamente difícil, visto que dependências complexas entre mudanças imprudentes anteriores vêm à tona

Evolução e Manutenção

- A abordagem centrada em arquitetura oferece uma base sólida para uma evolução eficaz:
  - Foco sustentado em um modelo arquitetural explícito, real e modificável
- Fases do processo de evolução:
  1) Motivação
  2) Avaliação
  3) Escolha e projeto da abordagem
  4) Execução, incluindo a preparação para a próxima rodada de adaptação

Evolução e Manutenção

- Motivação:
  - Dentre as diversas motivações para evolução, destaca-se a criação de novas versões de um produto
  - Justifica o estudo de famílias de produto
- Avaliação:
  - A mudança é examinada para determinar sua viabilidade. Caso seja viável, como ela será alcançada ?
  - Requer conhecimento aprofundado sobre o produto em questão
  - Se um modelo arquitetural fiel à implementação está disponível a compreensão e análise da mudança ocorrem de forma eficaz

Evolução e Manutenção

- Avaliação:
  - Se não existe modelo arquitetural ou o modelo não é consistente com a implementação pode-se utilizar engenharia reversa. Isso custa tempo e dinheiro
  - Erros de manutenção surgem de indevidas avaliações:
    - Se não há conhecimento suficiente sobre a estrutura existente os planos de modificação irão falhar, principalmente nos pontos desconhecidos
  - Um bom modelo arquitetural oferece uma base justificada para decidir se uma mudança desejada é razoável ou não
    - Pode-se verificar que a mudança é extremamente custosa ou que inviabiliza propriedades do sistema
    - Mantêm-se a integridade arquitetural e atenua a volatilidade dos requisitos

Evolução e Manutenção

- Escolha e projeto da abordagem:
  - A abordagem deve satisfazer todos os requisitos que motivaram a mudança
  - É realizada uma escolha entre alternativas
- Execução:
  - O primeiro artefato a ser modificado é o modelo da arquitetura
  - Só então mudanças no código-fonte são realizadas
  - A consistência deve ser sempre mantida. Ferramentas podem ajudar nesta tarefa
  - A manutenção não está concluída até que os modelos estejam consistentes

Processos de Desenvolvimento

- A arquitetura permeia todas as atividades de desenvolvimento e manutenção do software
- O conjunto de decisões arquiteturais é criado em consonância com os requisitos e continua se expandindo até a manutenção
- A natureza central da arquitetura é obscurecida nas caracterizações tradicionais das atividades de desenvolvimento de software:
  - As atividades do processo são o ponto central, não encontra-se a arquitetura em nenhum lugar
  - Exige limites rígidos entre os tipos de atividades

Processos de Desenvolvimento

- Afirmar que o processo deve ser centrado na arquitetura não quer dizer que existe uma única forma correta de realizar o desenvolvimento:
  - Estratégias particulares podem se beneficiar dos pontos fortes e preferências da organização
  - Diferentes ambientes de codificação irão demandar maior atenção a certas atividades
- Um bom formalismo para comparar e compreender diferentes estratégias deve disponibilizar um modo de ver as atividades mas também o papel central das arquiteturas

Processos de Desenvolvimento

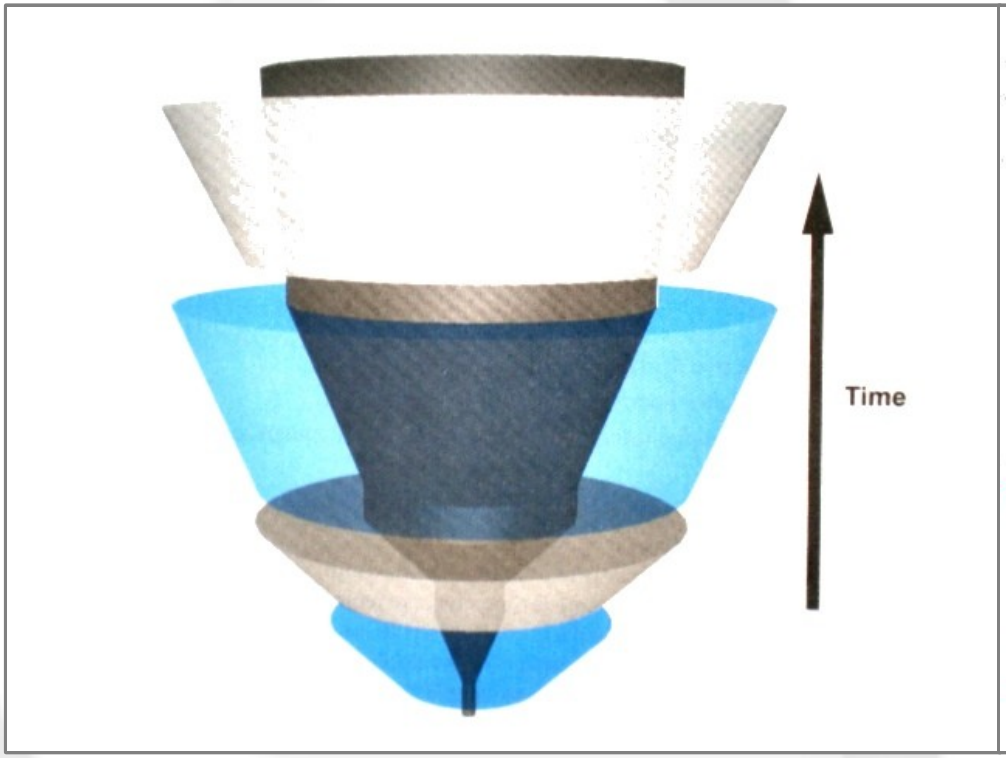
- Visualização em Turbina:
  - Método para ilustrar um conjunto integrado de atividades de desenvolvimento no qual é evidenciado o papel central da arquitetura do software
- Leva em consideração diferentes aspectos do desenvolvimento:
  - Tempo
  - Tipos de atividades ativas em um determinado tempo
  - Esforço (ex: horas trabalhadas) em um determinado tempo
  - Estado do produto (ex: conteúdo geral do projeto ou conhecimento acumulado)

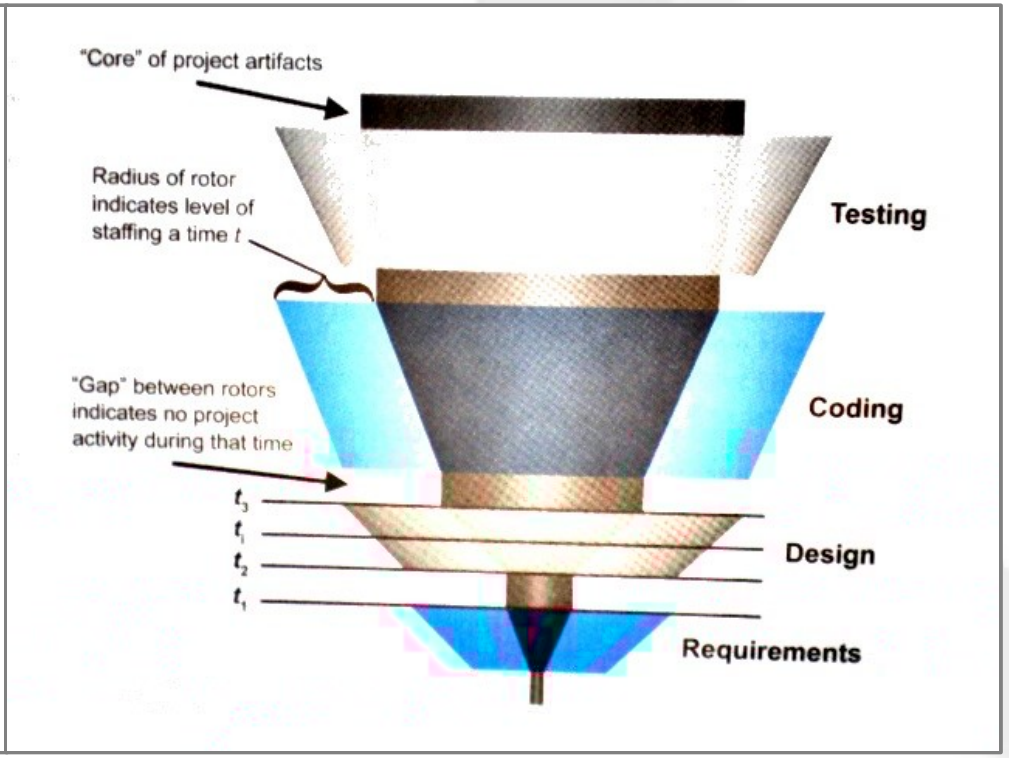
Processos de Desenvolvimento

- Visualização em Turbina:
  - É definida espacialmente por um conjunto de anéis com largura e espessura variáveis, dispostos ao longo de um núcleo
  - O eixo do núcleo é o tempo
  - O núcleo representa o produto sendo desenvolvido
  - Os anéis representam fases delimitadas pelo tempo
  - A espessura do anel denota a duração das (possivelmente concorrentes) atividades do anel
  - O volume do anel representa o investimento realizado
  - Secções transversais de um mesmo anel podem ser diferentes e intervalos entre anéis podem existir

Processos de Desenvolvimento

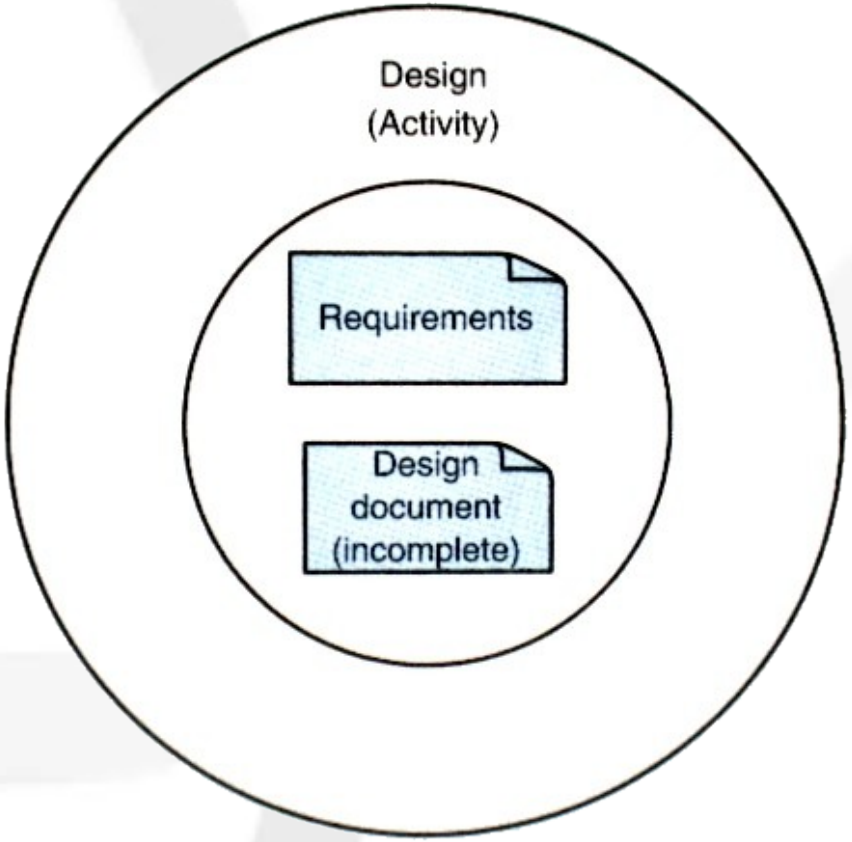
- Visualização em Turbina (modelo em cascata):





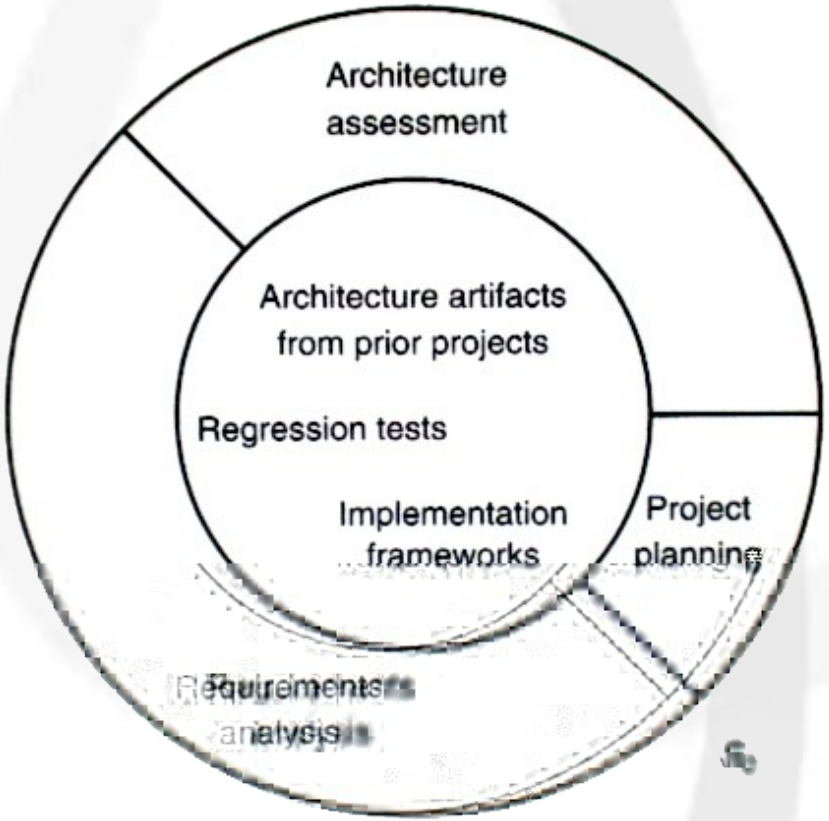
Processos de Desenvolvimento

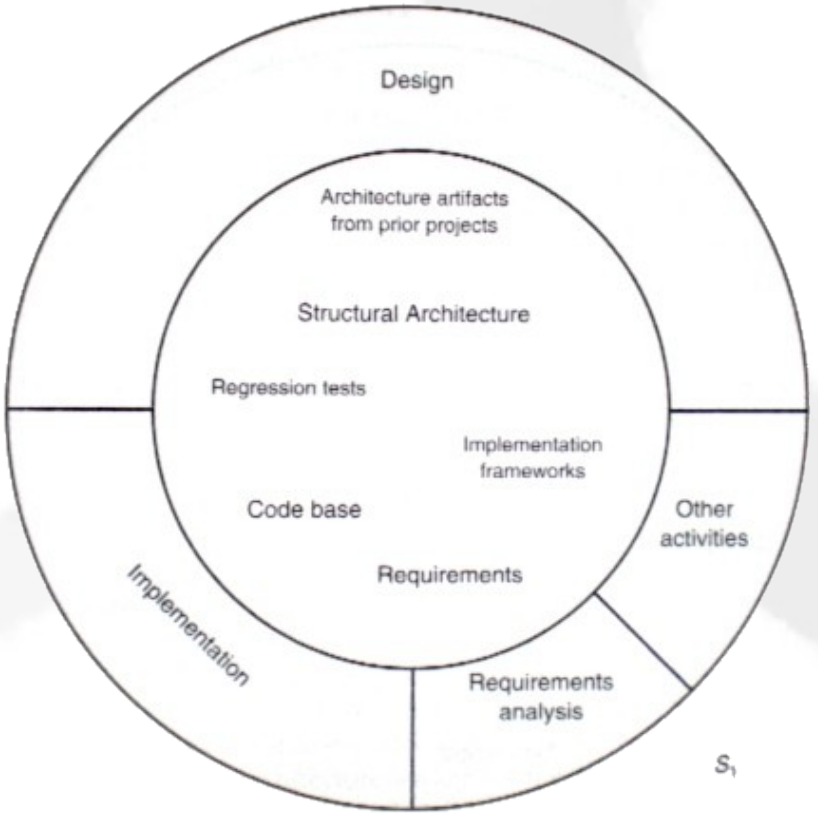
- Visualização em Turbina (modelo em cascata):
  - Secção transversal no tempo t_1



Processos de Desenvolvimento

- Visualização em Turbina (atividades concorrentes):
  - Secções transversais nos tempos t0 e t1



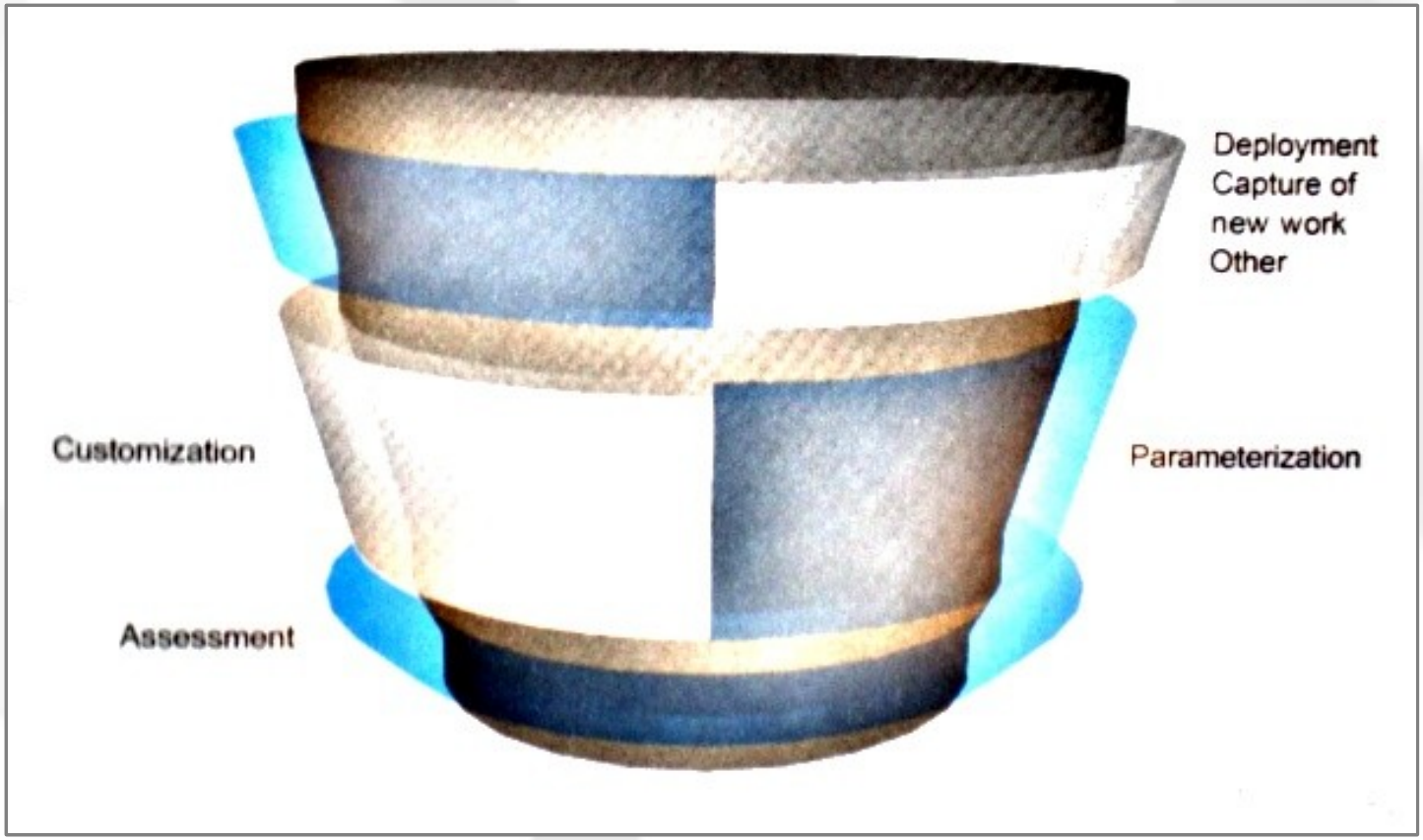


Processos de Desenvolvimento

- Visualização em Turbina (atividades concorrentes):
  - Secção transversal próximo ao fim do projeto

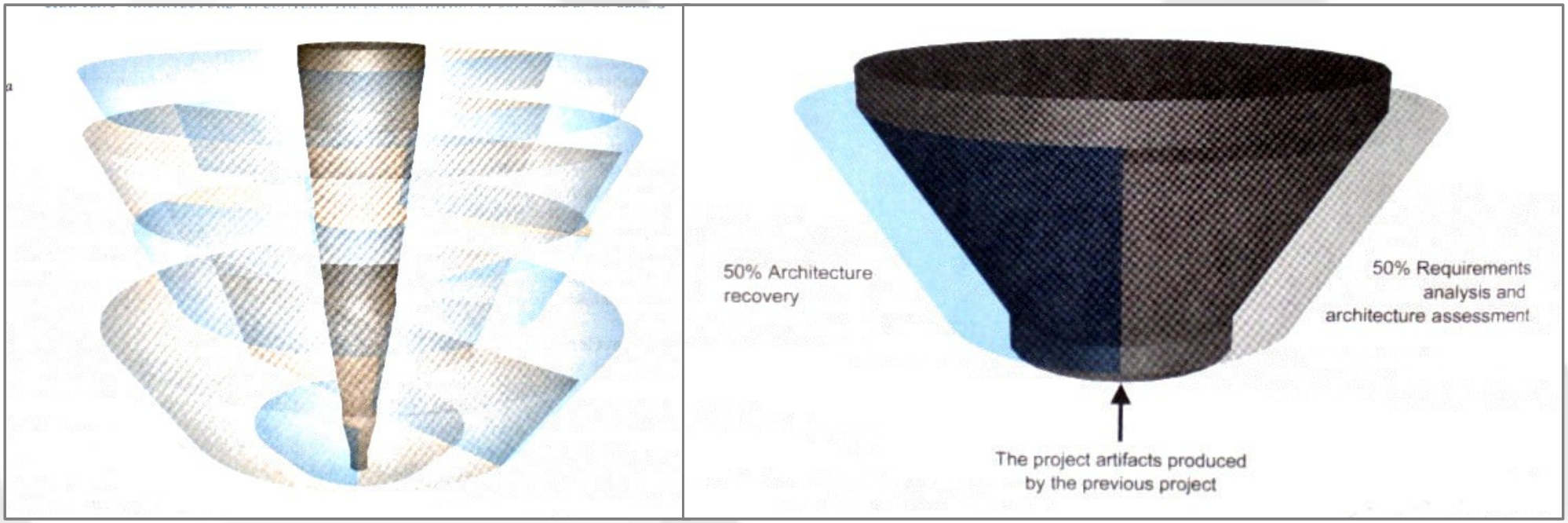
Processos de Desenvolvimento

- Exemplo de descrição de processo:
  - Domain-Specific Software Architecture (DSSA)



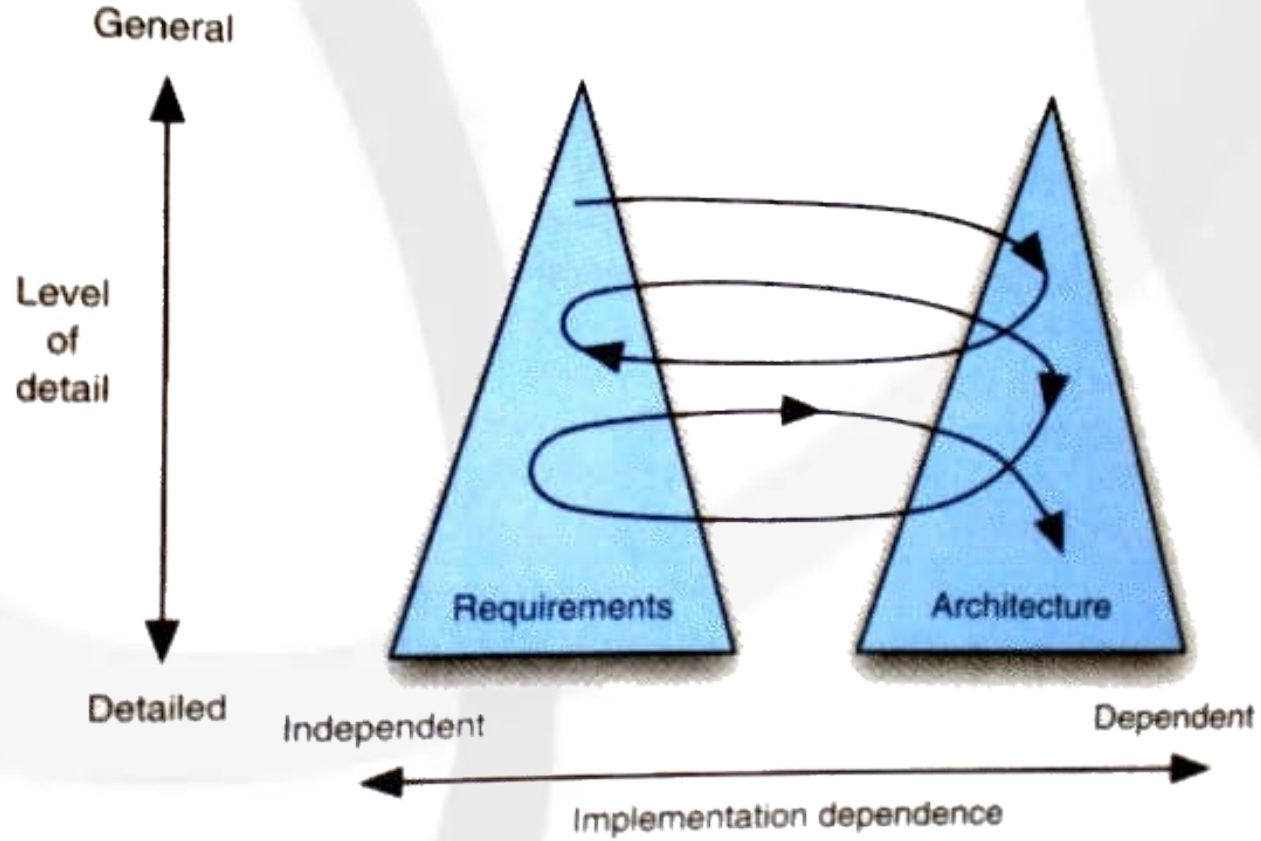
Processos de Desenvolvimento

- Exemplo de descrição de processo:
  - Desenvolvimento ágil



Processos de Desenvolvimento

- Outros modelos de processo:
  - Twin Peaks [Nuseibeh 2001]



Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software Conceitos Básicos

Sandro S. Andrade
sandroandrade@ifba.edu.br

Arquitetura de Software

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

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

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:

> 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:

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

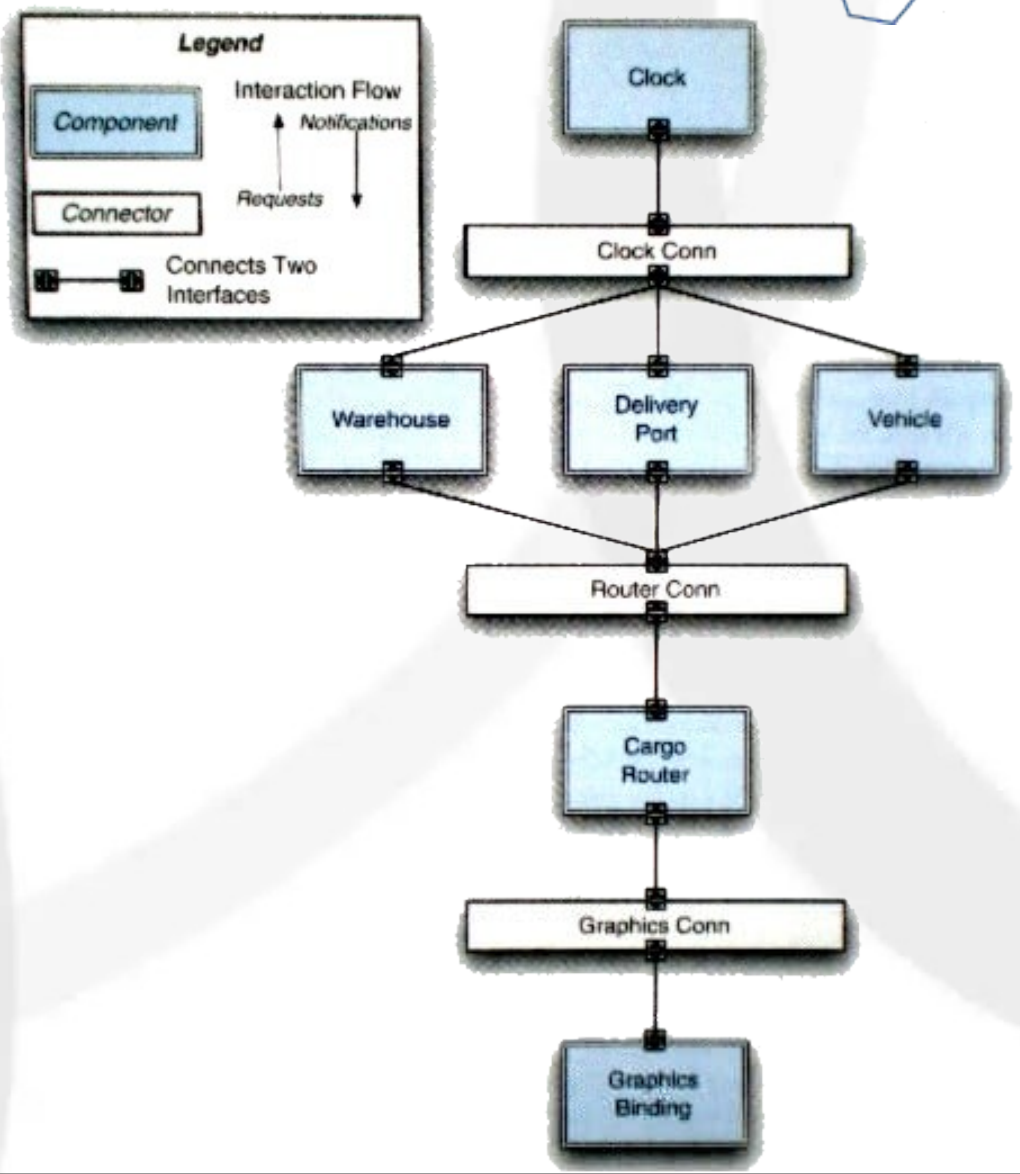
- Representa a arquitetura "implementada" do sistema
- No início do projeto (tempo t1) tem-se a criação do conjunto P1 e os conjuntos A1 e D1 podem estar vazios (desenvolvimento greenfield)
- No desenvolvimento brownfield, onde um conjunto de artefatos implementando parcialmente a arquitetura já existe, A0 e D0 são não-vazios enquanto P0 é vazio

Arquitetura Descritiva

- P0 também pode estar vazio e D0 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

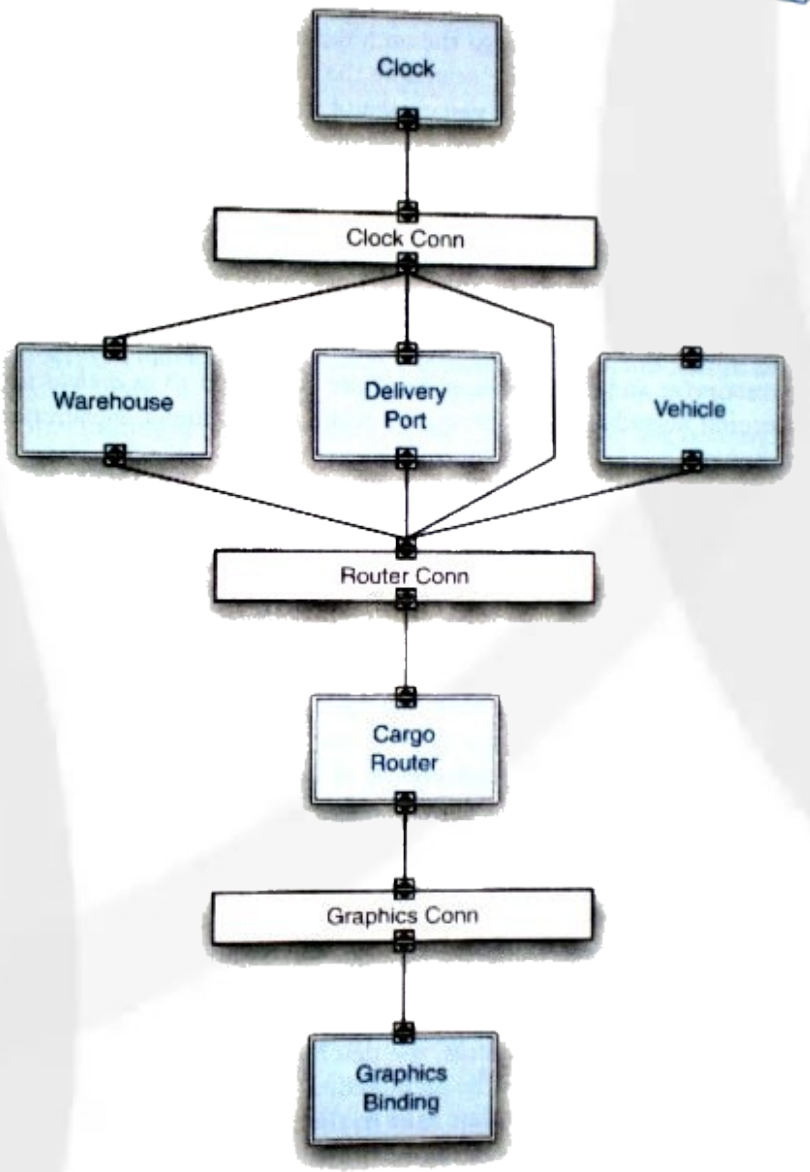


Arquitetura Prescritiva e Descritiva

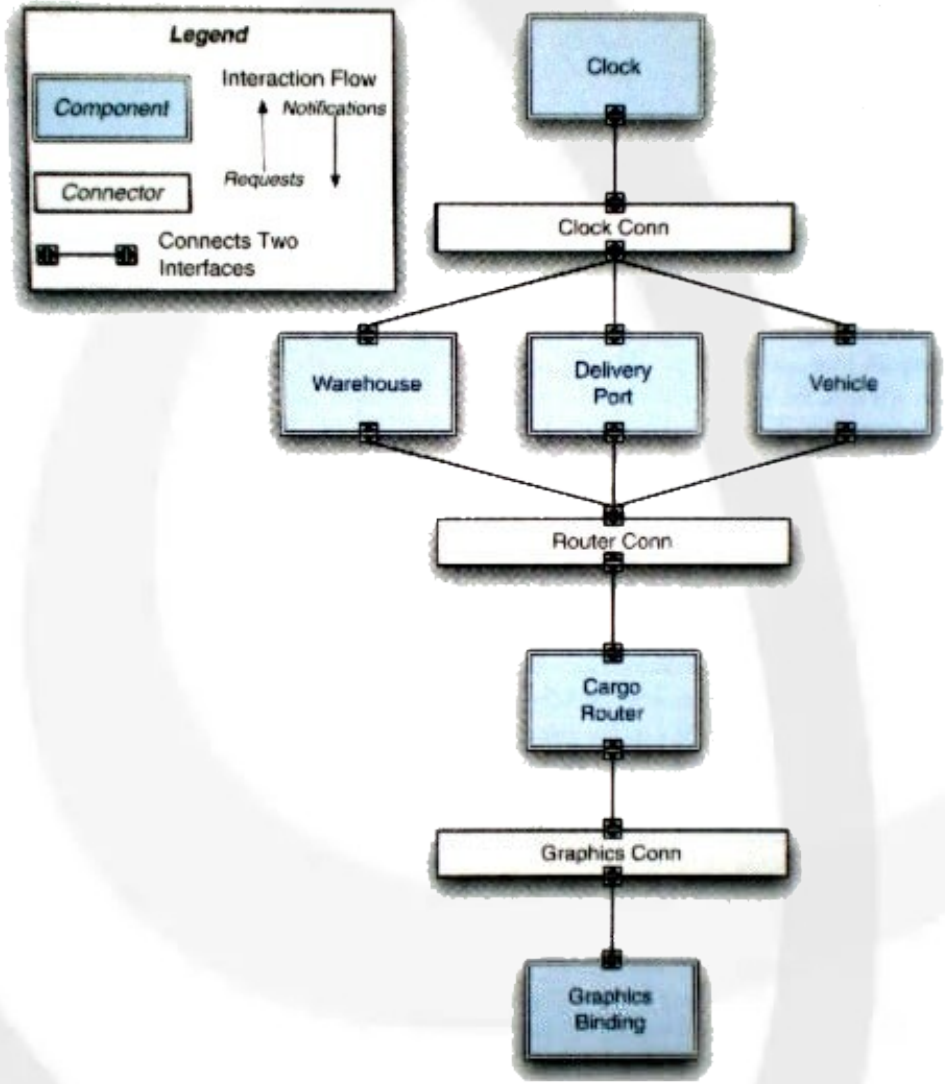
- O exemplo é útil porque:
  - Sua simplicidade sugere que seus arquitetos puderam avaliar todas as decisões arquiteturais e seus trade-offs antes da implementação da aplicação
  - A arquitetura foi implementada com a ajuda de um architectural framework, permitindo implementar todas as decisões diretamente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

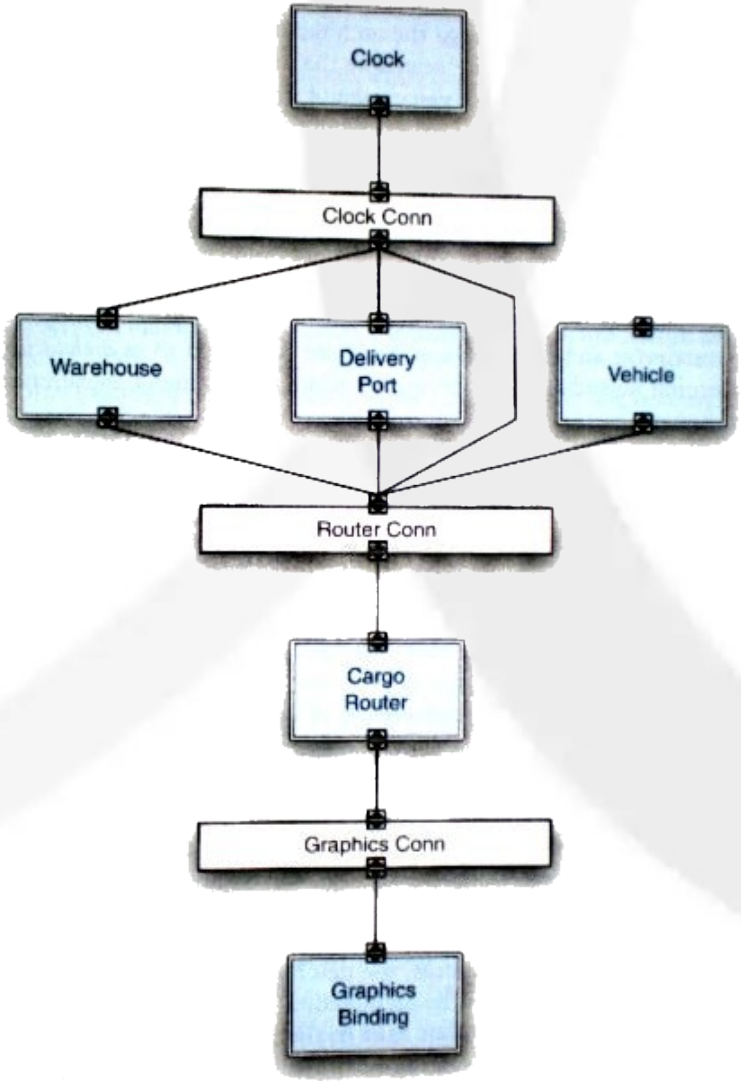
Arquitetura Prescritiva e Descritiva

- 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

discrepância existente entre as arquiteturas prescritiva e descritiva do sistema

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 nen#uma das decisões de projeto da arquitetura prescritiva

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

Perspectiva Arquitetural

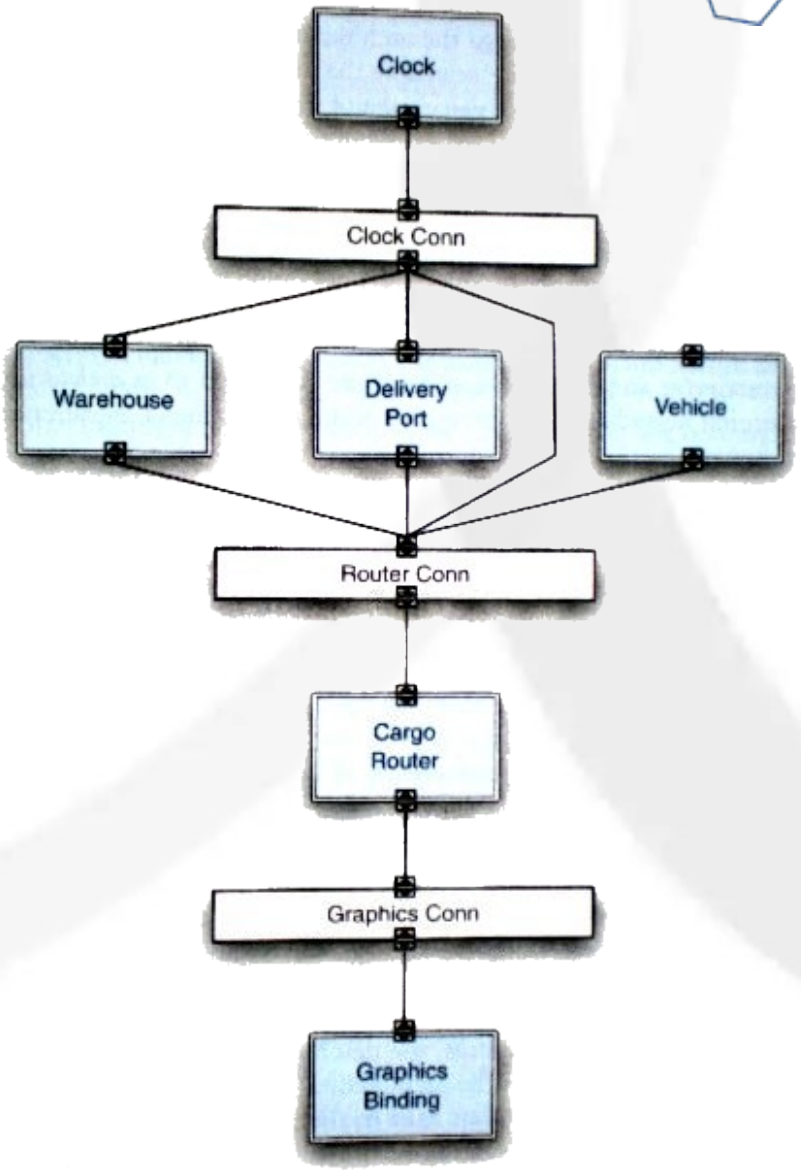
- Perspectiva (visão) arquitetural:
  - Tem como objetivo destacar determinados aspectos de uma arquitetura ao mesmo tempo em que omite outros

um conjunto não$va%io 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

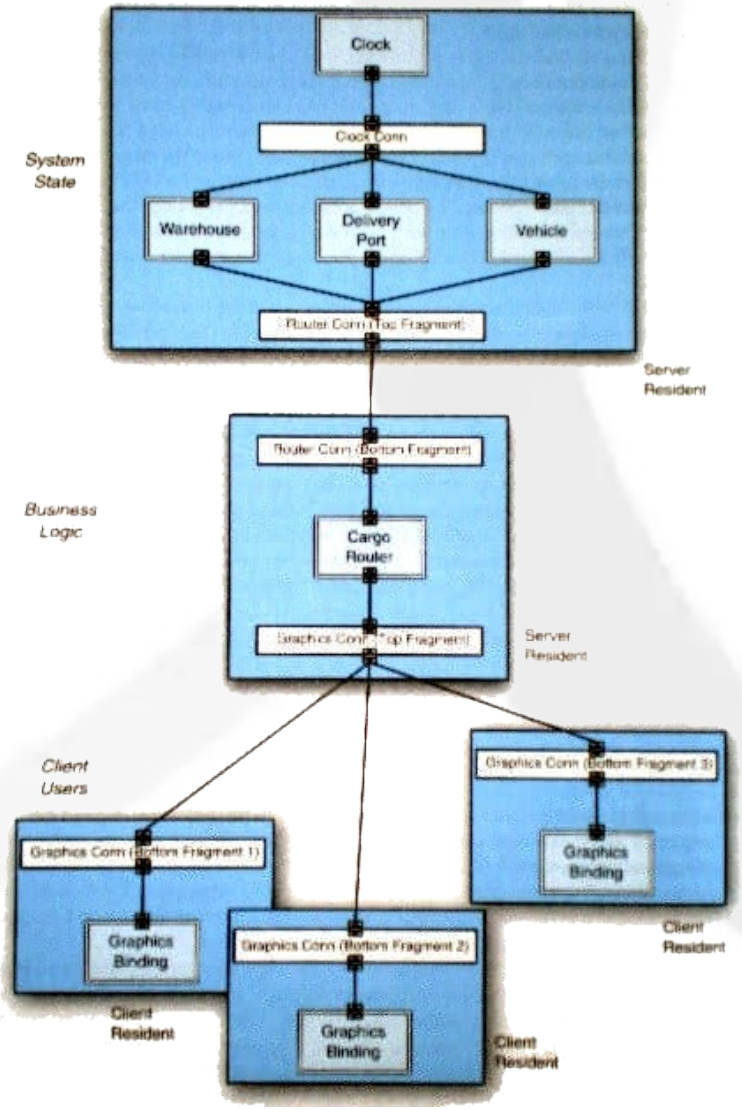
Perspectiva Arquitetural

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



Perspectiva Arquitetural

- 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

& ***elements*** 'de processamento, dados ou conexão",
form, ***rationale*** 'intenções, pressupostos, escol#as sutis, restrições externas, estilos e padrões adotados, etc" (

)*err+ , - olf $. / /01

a organi%ação fundamental do sistema, implementada por seus componentes, os relacionamentos entre eles e deles com o am2iente, e os princípios que governam seu projeto e evolução

)3456768885tandard .9:.$0;;;1

de um sistema j< implantado determinada pelos seus aspectos mais difíceis de serem modificados

)=#ris >er#oef $ 0;;?1

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:

uma entidade arquitetural que! *1*" encapsula um su2conjunto das funcionalidades e7ou dados do sistema@ *2*" restringe o acesso a este su2conjunto atrav s de interfaces explicitamente definidas@ e *3*" possui dependAncias $ explicitamente definidas B 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:

unidade de composição formada somente de interfaces definidas de forma contratual e dependAncias explícitas de contexto

)5%+persCi $. / / :1

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

elemento arquitetural respons<vel 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 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

coleção identificada de decisões arquiteturais de projeto
que! *1*" são aplic<veis a um determinado contexto de desenvolvimento@ *2*"
restringe as decisões arquiteturais específicas de um sistema em particular
dentro deste contexto@ e *3*" indu% qualidades 2en ficas 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

coleção identificada de decisões arquiteturais de projeto que são aplic<veis a um pro2lema recorrente de desenvolvimento e parametri%adas de modo a serem aplicadas em qualquer contexto de desenvolvimento de *software* no qual o pro2lema aparece

Padrão Arquitetural

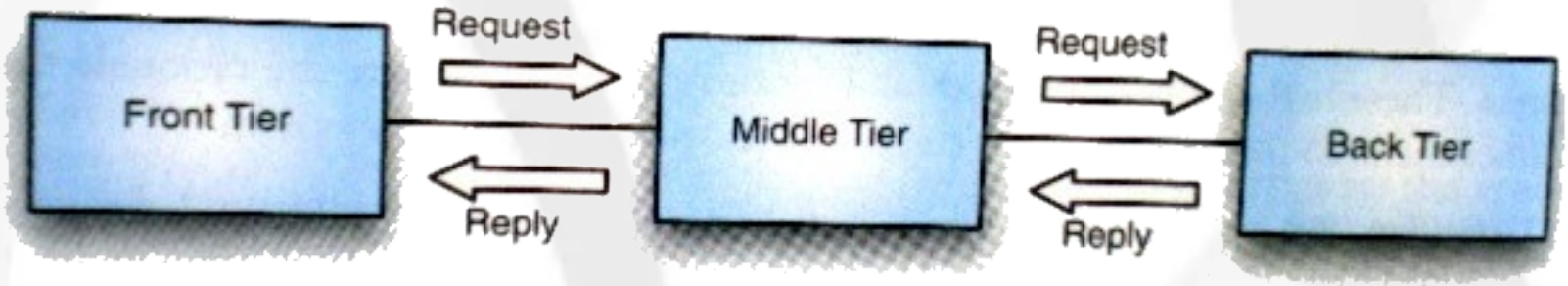
- As definições de estilo e padrão arquitetural são parecidas, nem sempre poderemos fazer a distinção com clareza
- Porém, estilos e padrões diferem em alguns aspectos
  - Escopo: estilos se aplicam a um contexto de desenvolvimento (ex: sistemas altamente distribuídos, sistemas GUI-intensive), enquanto 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 ? request-triggered ? single-request-single-reply ?)

Padrão Arquitetural

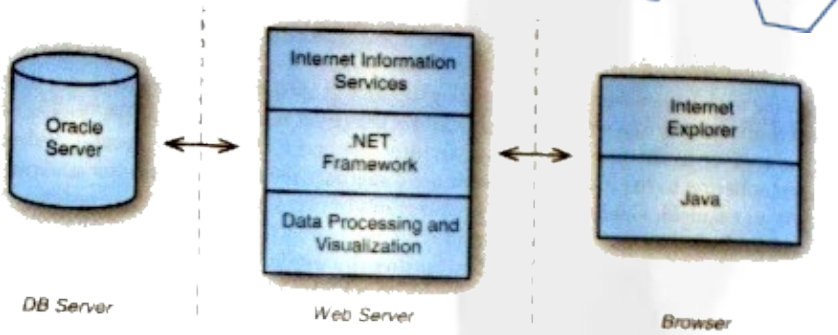
- Exemplo de padrão arquitetural:
  - Sistema em três camadas: parâmetros
    - Quais facilidades para interface de usuário, processamento, armazenamento e acesso a dados – específicos da aplicação – são necessários ?
    - Como elas devem ser organizadas dentro de cada camada ?
    - Quais mecanismos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

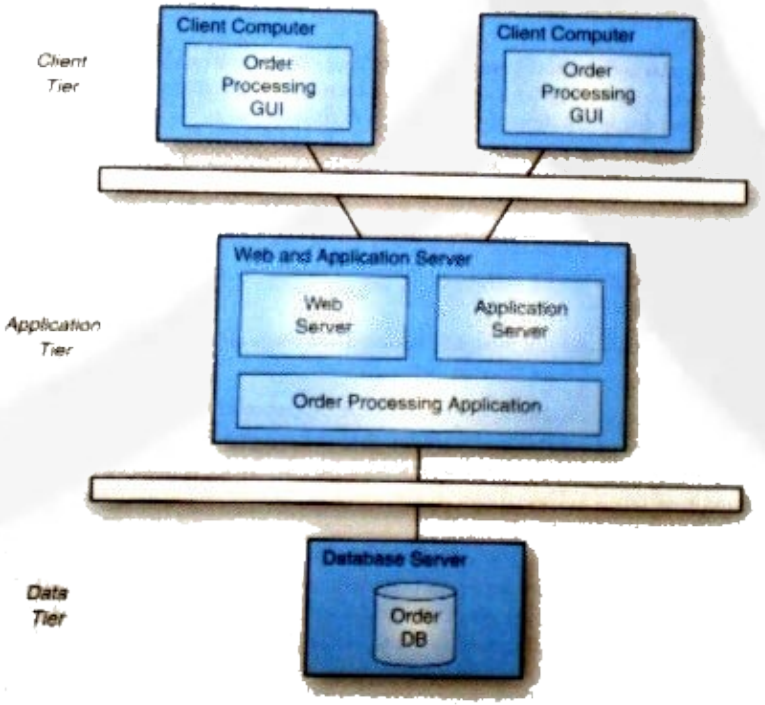
Padrões x Estilos

- 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 Arquiteturais

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

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

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 Arquiteturais

uma linguagem ou um meio de captura
das decisões arquiteturais de projeto

- As notações podem ser textuais ou gráficas, informais (diagramas em slides), semi-formais (UML), formais (ADL's), 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:

> 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

Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software Projeto Arquitetural

Sandro S. Andrade
sandroandrade@ifba.edu.br

Projeto Arquitetural

- Como os projetos arquiteturais são criados ?
- Qual abordagem é ensinada aos engenheiros e arquitetos ?
- Alguns acreditam que projeto arquitetural não pode ser ensinado como um método.“Observe o mestre e aprenda por osmose …“
- Outros acreditam ser um dom com o qual você nasce ou não
- Discorda-se dessas duas visões

Processo de Projeto

- Fases (Jones, 1970):
  - Viabilidade: identificação de um conjunto de conceitos viáveis. Um conjunto de arranjos alternativos para o projeto é rapidamente derivado
  - Projeto Preliminar: seleção e desenvolvimento do melhor conceito, o qual é dividido e processado, em paralelo, nas próximas fases
  - Projeto Detalhado: desenvolvimento das descrições de engenharia do conceito
  - Planejamento: avaliação e alteração do conceito de modo a adequá-lo aos requisitos de produção, distribuição, consumo e descarte

Processo de Projeto

- O modelo de Jones é tão amplamente utilizado que é difícil perceber que outras abordagens são possíveis:
  - Ele pode não funcionar sempre ou, mesmo funcionando, não produzir os melhores resultados:
    - Impossibilidade de identificação do conjunto de conceitos viáveis para a estrutura geral do sistema (fase 1)
    - A fase 1 é geralmente realizada por um único arquiteto. Para sistemas grandes e complexos torna-se difícil. Utilizar uma equipe de arquitetos traz novos problemas
    - De forma similar, a abordagem não é adequada quando deseja-se projetar uma arquitetura para um conjunto de produtos. Novamente tem-se maior complexidade

Processo de Projeto

- Diversas estratégias de projeto (modelos de processo aplicados no nível de projeto):
  - Padrão: modelo linear de Jones
  - Cíclico: se problemas ou abordagens inviáveis são encontrados nas fases 2-4 o processo retorna a uma fase anterior
  - Paralelo: após a (ou na) fase 1 alternativas independentes são exploradas em paralelo
  - Adaptativo: a estratégia de projeto a ser utilizada na próxima fase é decidida no fim da fase atual, com base em insights obtidos nesta fase
  - Incremental: projeta-se à medida em que o desenvolvimento é realizado, melhorando de forma incremental qualquer projeto existente após a fase anterior

Concepção Arquitetural

- Como identificar um (ou vários) arranjos viáveis para o projeto ?
  - Resposta fácil 1: aplique as ferramentas fundamentais de projeto ensinadas pela Engenharia de Software: abstração e modularidade
    - São necessárias e úteis mas são apenas ferramentas. Onde e como manejá-las ?
  - Resposta fácil 2: inspiração …
    - Criatividade é necessário mas não é uma varinha mágica
    - Pode-se minimizar e isolar as partes do projeto que requerem maior criatividade e aplicar técnicas mais prosaicas e previsíveis nas outras partes
    - Precisa-se saber, entretanto, onde é necessário a criatividade e onde não é

Concepção Arquitetural

- Como identificar um (ou vários) arranjos viáveis para o projeto ?
  - Resposta mais comum, efetiva e apropriada: use sua experiência aplicada
  - Não é uma resposta superficial, existe uma sofisticação considerável no uso de experiência
  - Não é infalível e as vezes não é suficiente, entretanto

Concepção Arquitetural

- Ferramentas conceituais fundamentais:
  - Separação de concerns
  - Abstração
  - Modularidade
  - Antecipação de mudanças
  - Projeto voltado para generalidade

Concepção Arquitetural Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Abstração é a seleção de um conjunto de conceitos como representantes de um todo mais complexo
  - Pode ser bottom-up, partindo dos detalhes e chegando aos conceitos sumários
  - Entretanto, quando relacionado a projeto, geralmente é aplicado top-down: define-se um arranjo de partes abstratas (conceitos) que compõem a solução ainda em alto nível e então reifica-se estes conceitos em estruturas mais completas até chegar no código-fonte
  - Poderia ser usado os termos refinamento ou dedução, porém utiliza-se abstração pois ela será uma caracterização precisa e útil do código-fonte

Concepção Arquitetural Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Mas a pergunta persiste:"Quais conceitos devem ser escolhidos no início de um projeto ?"
  - Encontre uma "máquina" simples que serve como abstração de um potencial sistema que irá realizar a tarefa desejada
    - Ex: planilha de cálculo: grafo de relacionamentos onde, quando um nó é modificado, os relacionamentos são todos re-avaliados para manter a planilha sincronizada
    - Ex: software para máquina de fax: uma máquina de estados pequena
    - Ex: sistema embarcado em aviões: lê-se os sensores, calcula-se as leis de controle, envia valores para os displays e atuadores e então repete-se tudo novamente

Concepção Arquitetural Ferramentas Fundamentais

- Abstração e Máquinas Simples:
  - Embora possam parecer triviais ou inadequadas elas apresentam uma primeira concepção plausível sobre como a aplicação poderia ser construída

Domain	Simple Machines
Graphics	Pixel arrays Transformation matrices Widgets Abstract depiction graphs
Word processing	Structured documents Layouts
Industrial process control	Finite state machines
Income tax return preparation	Hypertext Spreadsheets Form templates
Web pages	Hypertext Composite documents
Scientific computing	Matrices Mathematical functions
Financial accounting	Spreadsheets Databases Transactions

Concepção Arquitetural Ferramentas Fundamentais

- Separação de concerns:
  - Sub-divisão do problema em partes independentes
  - Ex: a interface de usuário de uma ATM é independente da lógica utilizada para controlar as operações bancárias
  - Se os conceitos são inerentemente independentes a separação é trivial
  - É mais difícil quando os problemas estão aparentemente (ou de fato) entrelaçados
  - Estes entrelaçamentos ocorrem por:
    - Abuso de linguagem: termo único (ex: conta) descrevendo diferentes partes da aplicação
    - Questões de eficiência: ex: surrogate keys em bancos de dados

Concepção Arquitetural Ferramentas Fundamentais

- Separação de concerns:
  - Um exemplo importante é a divisão da estrutura do sistema em componentes (loci de computação) e conectores (loci de comunicação)
  - Frequentemente envolve muitos trade-offs
  - Independência total de concerns pode ser impossível, requerendo uma análise dos trade-offs de desempenho, custo, aparência e funcionais das concepções alternativas

Concepção Arquitetural Experiência Refinada

- Ao mesmo tempo em que as ferramentas fundamentais são de uso inegável geralmente representam um guia insuficiente para o projetista
- Experiência pode ser um poderoso aliado para manejar as ferramentas fundamentais de modo efetivo
- Realizar projetos por si só não produz a maturidade necessária para lidar com novos problemas
- É necessário uma reflexão e refinamento posteriores de modo a compreender os problemas essenciais e lições aprendidas

Concepção Arquitetural Experiência Refinada

- A meta é a escolha criteriosa de técnicas apropriadas ao contexto em questão
- Inclui análise de lições de sucesso mas também de falhas
- As lições não se restringem àquelas do próprio projetista:
  - Ampla leitura técnica e investigação de boas soluções
  - Conferências técnicas
- Não há virtude alguma em reinventar a roda

Concepção Arquitetural Experiência Refinada

- Quando se possui experiência técnica:
  - É mais fácil encontrar o conjunto inicial de arranjos viáveis alternativos
  - Não se gasta tempo em pontos para os quais já existe uma solução validada
  - Facilita a comunicação na equipe de desenvolvimento
- Entretanto não é livre de riscos:
  - "Quando se está com o seu martelo favorito tudo passa a se parecer com um prego". Gera-se sistemas que funcionam, mas deficientes ou sub-ótimos
  - As limitações também são importadas
  - Inibe a inovação

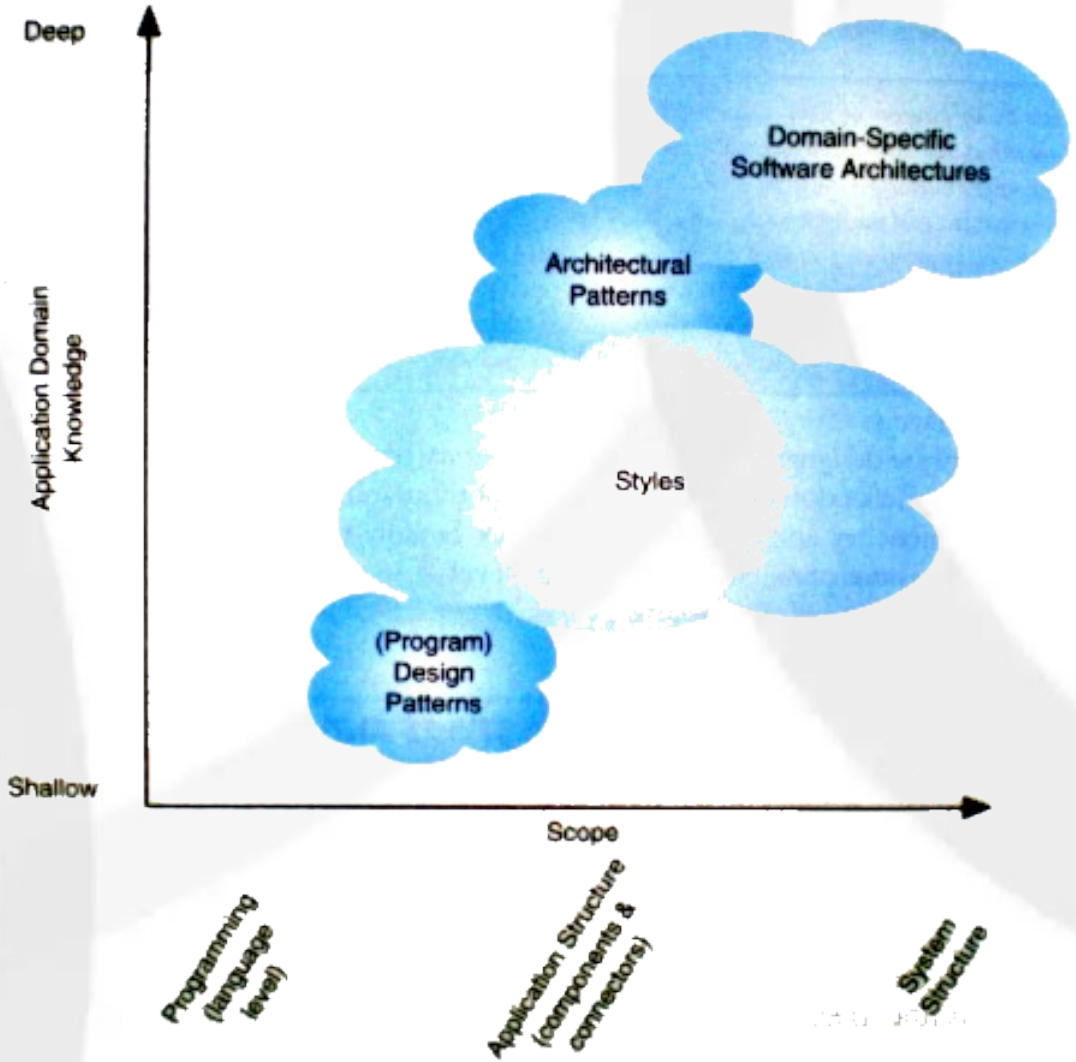
Estilos e Padrões Arquiteturais

são projetados para capturar o con#ecimento de projetos efetivos no alcance de metas específicas dentro de um contexto particular de aplicações

- Exemplo na construção civil:
  - Uma determinada combinação de metas e contexto pode demandar a construção de uma casa para uma família pequena numa região de clima mediterrâneo utilizando pedras e telhas como materiais básicos
  - O estilo apropriado seria Single-Story Villa
- Exemplo em software:
  - Metas e contexto demandam o desenvolvimento de um sistema de instant messaging operando entre sites remotos de uma empresa
  - O estilo apropriado seria Client-Server

Estilos e Padrões Arquiteturais

Estilos e padrões podem ser caracterizados tanto para problemas pequenos quanto mais complexos



Estilos e Padrões Arquiteturais

- As bordas do diagrama anterior não são precisamente definidas
- O eixo scope não é genuinamente linear ou totalmente ordenado
- O que um arquiteto denomina Padrão Arquitetural pode ser chamado de Estilo Arquitetural por outro

Pós-Graduação em Computação Distribuída e Ubíqua

INF612 - Aspectos Avançados em Engenharia de Software
Arquitetura de Software

Sandro S. Andrade
sandroandrade@ifba.edu.br