INF016 – Arquitetura de Software

02 – Reorientação da Engenharia de Software

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

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

- Esta ideia não é nova:

> ***"Na análise nós começamos pelo que é requerido, o tomamos como certo e traçamos consequências até que alcancemos um ponto de onde podemos começar a síntese"***

Pappus, século 4 D.C.

- Por síntese aqui entende-se o projeto ou elaboração da solução

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 ajudam 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

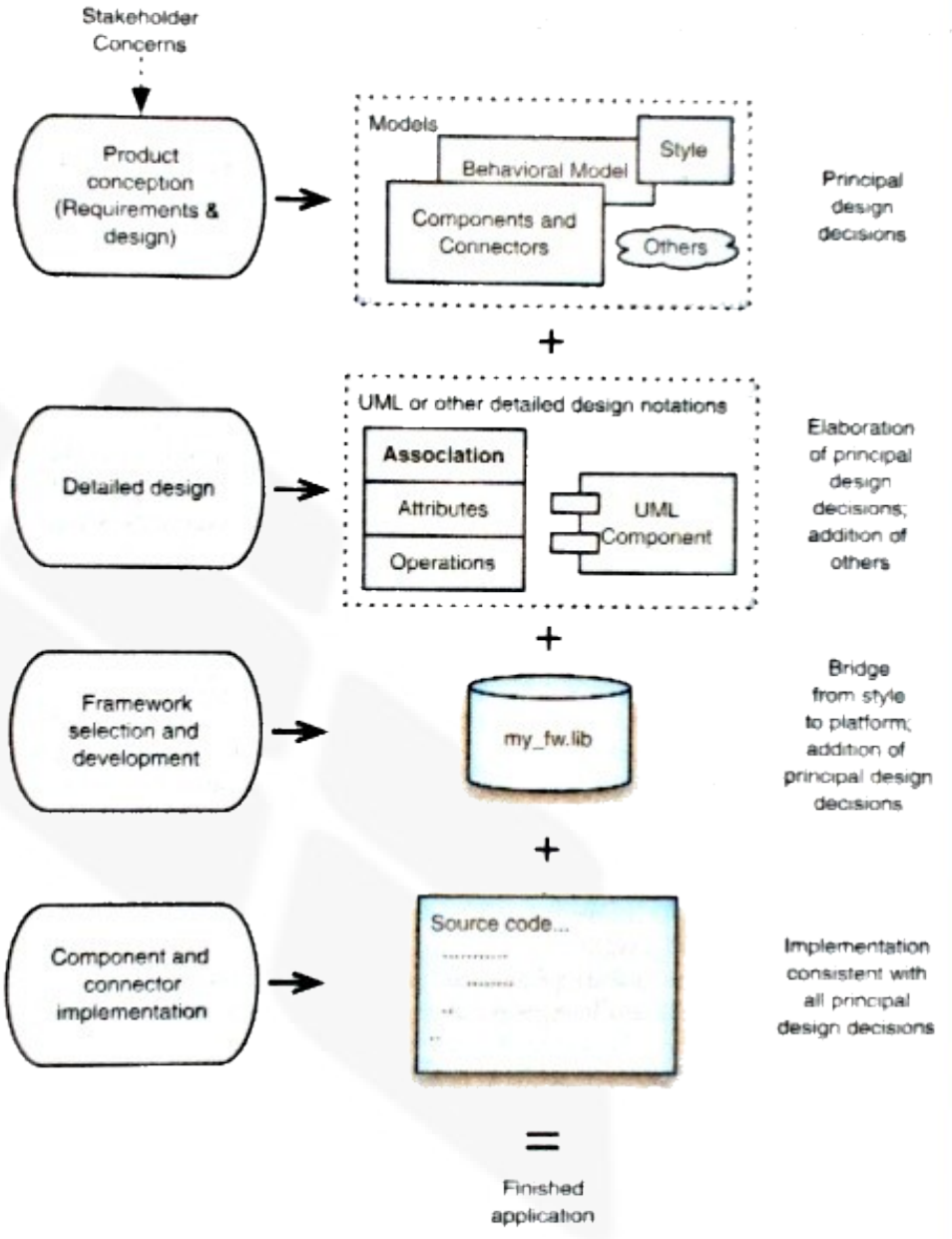
- Na prática essa definição não é totalmente adequada:
  - Uso de bibliotecas reduzem o custo e aumentam a qualidade, porém contêm funções e interfaces que não estão presentes na arquitetura
  - Se uma biblioteca barata e de qualidade implementa 98% da funcionalidade desejada e se as consequências da 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):
  1) Técnicas generativas: a implementação é gerada automaticamente e possui alta qualidade
     - Geralmente aplicada em domínios muito específicos
  2) 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*
  3) 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 produtos

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

1) Motivação:
   - Dentre as diversas motivações para evolução, destaca-se a criação de novas versões de um produto
   - Justifica o estudo de famílias de produto
2) 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

2) 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

3) Escolha e projeto da abordagem:
   - A abordagem deve satisfazer todos os requisitos que motivaram a mudança
   - É realizada uma escolha entre alternativas
4) 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

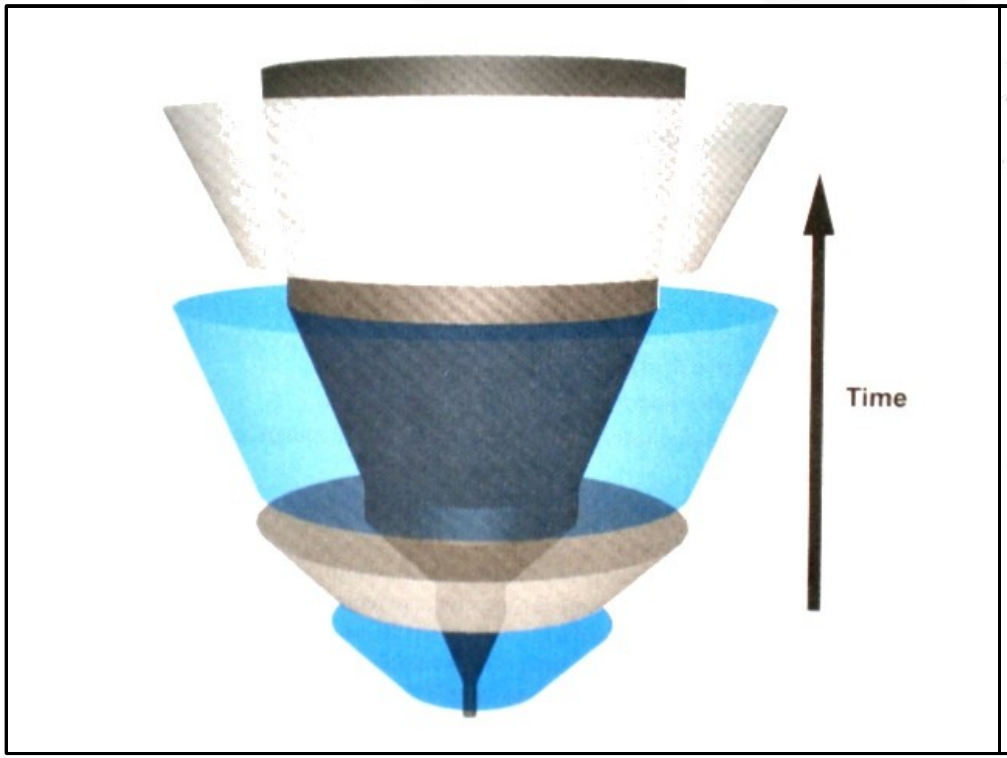
- Visualização em Turbina:
  - Método para ilustrar um conjunto integrado de atividades de desenvolvimento no qual é evidenciado o papel central da arquitetura do *software*
- Leva em consideração diferentes aspectos do desenvolvimento:
  - Tempo
  - Tipos de atividades ativas em um determinado tempo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

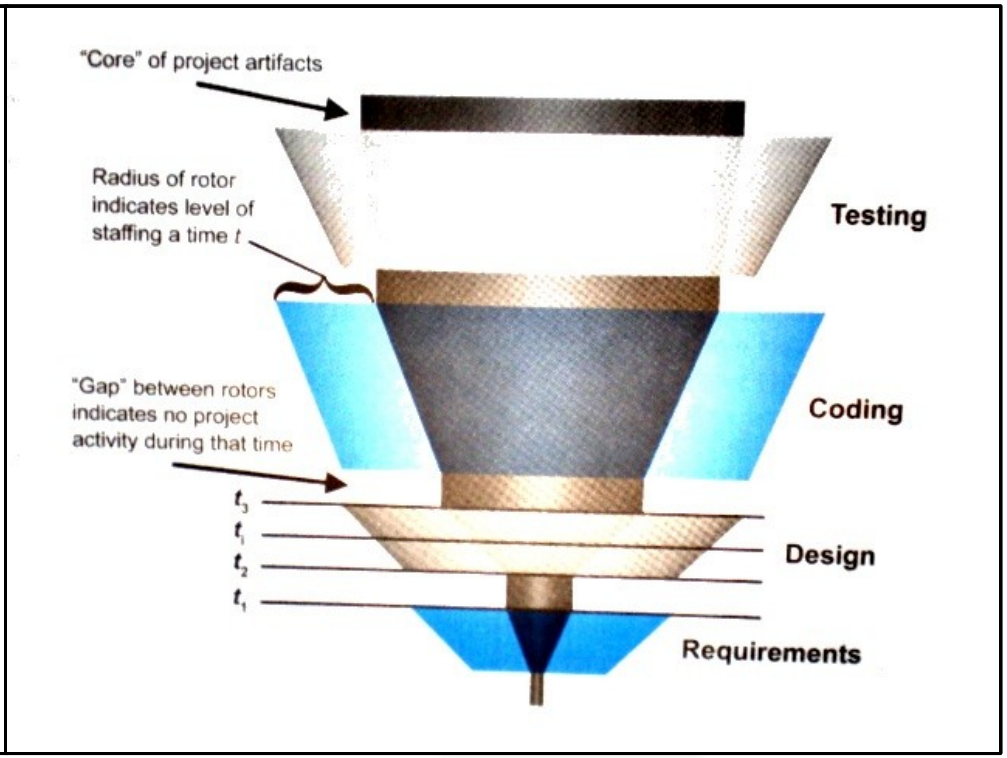
Processos de Desenvolvimento

- 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

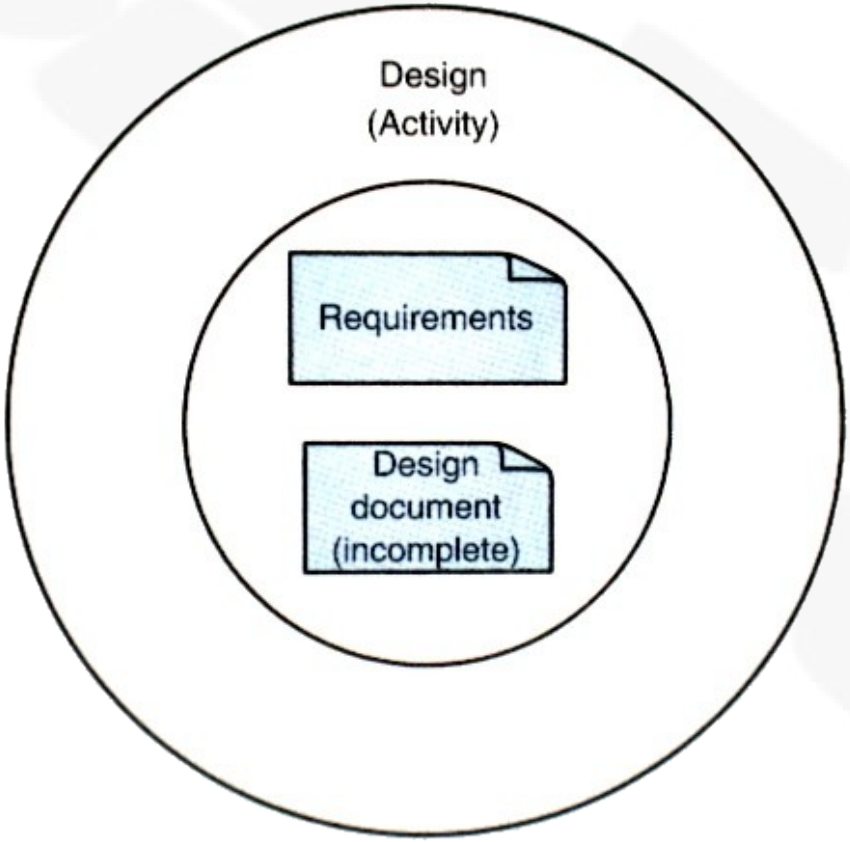
- Visualização em Turbina (modelo em cascata):





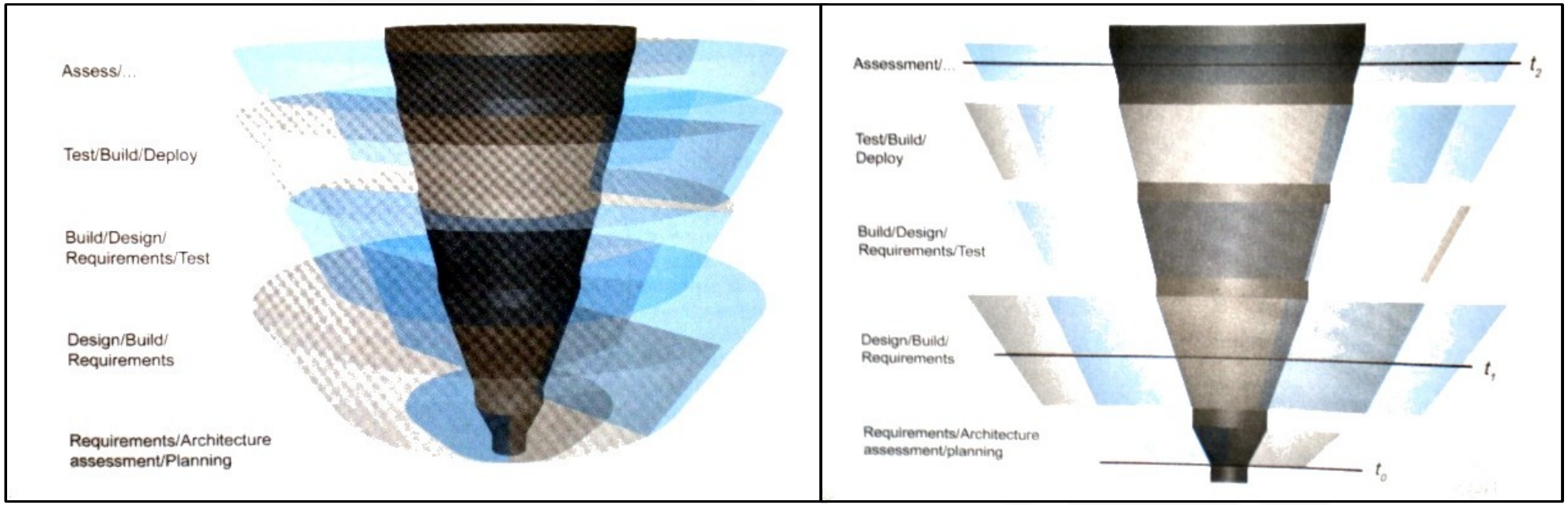
Processos de Desenvolvimento

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



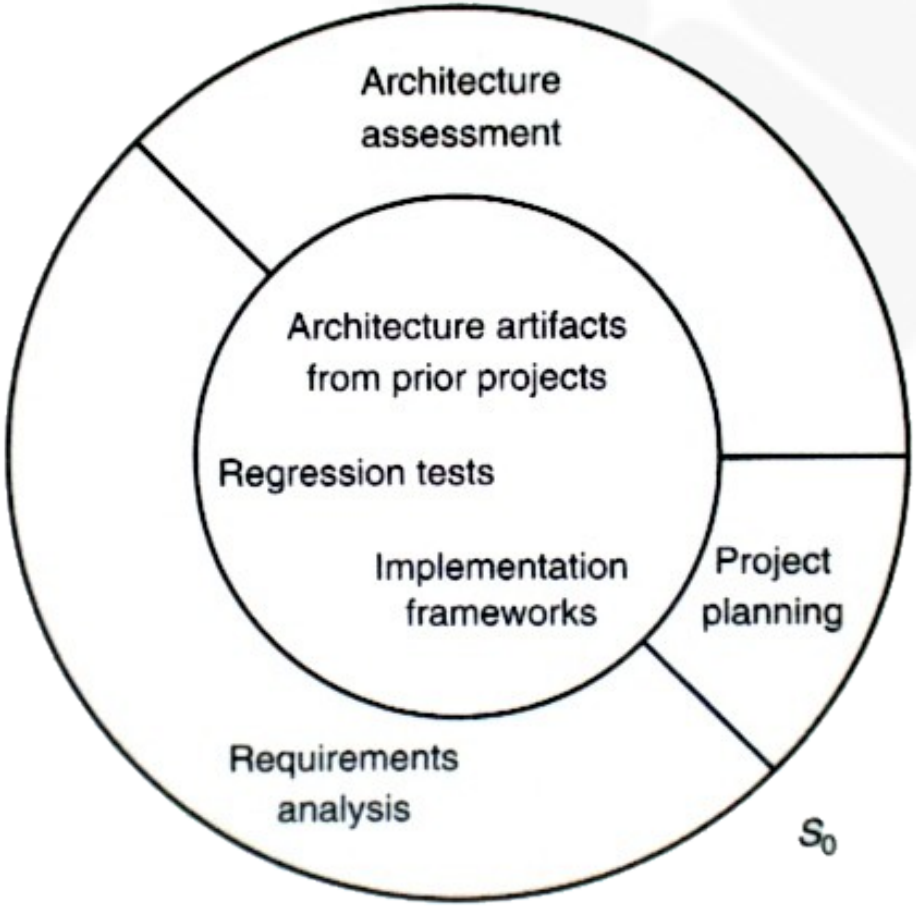
Processos de Desenvolvimento

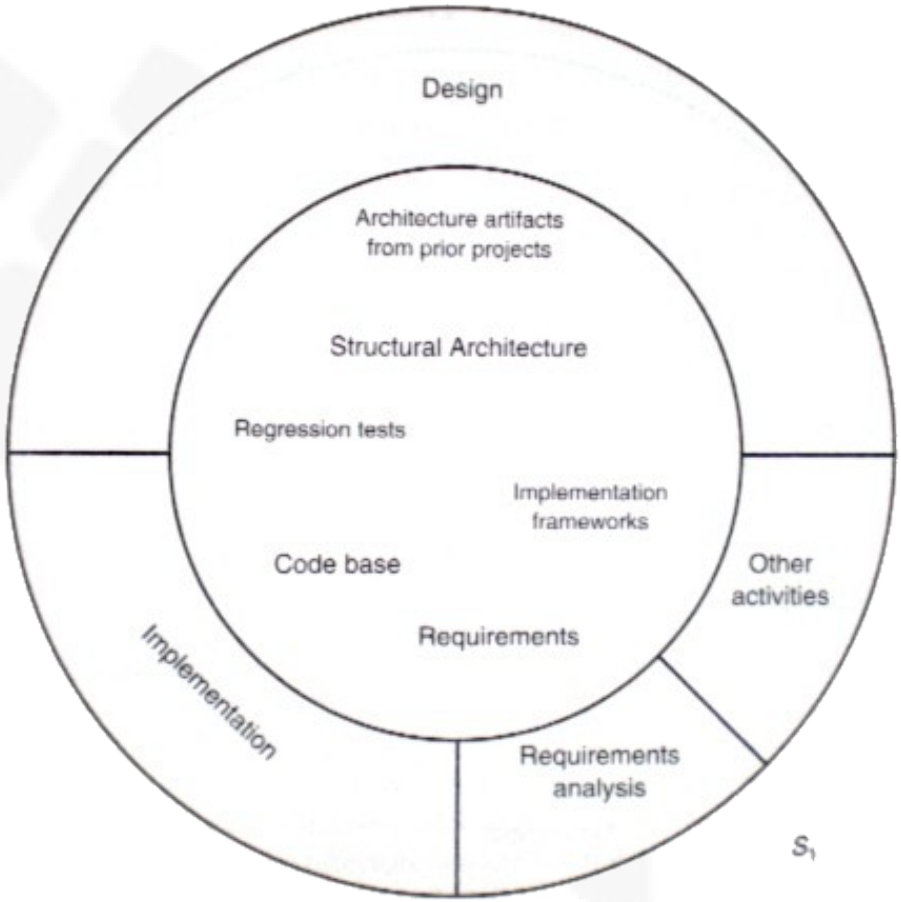
- Visualização em Turbina (atividades concorrentes):



Processos de Desenvolvimento

- Visualização em Turbina (atividades concorrentes):
  - Secções transversais nos tempos t_0 e t_1



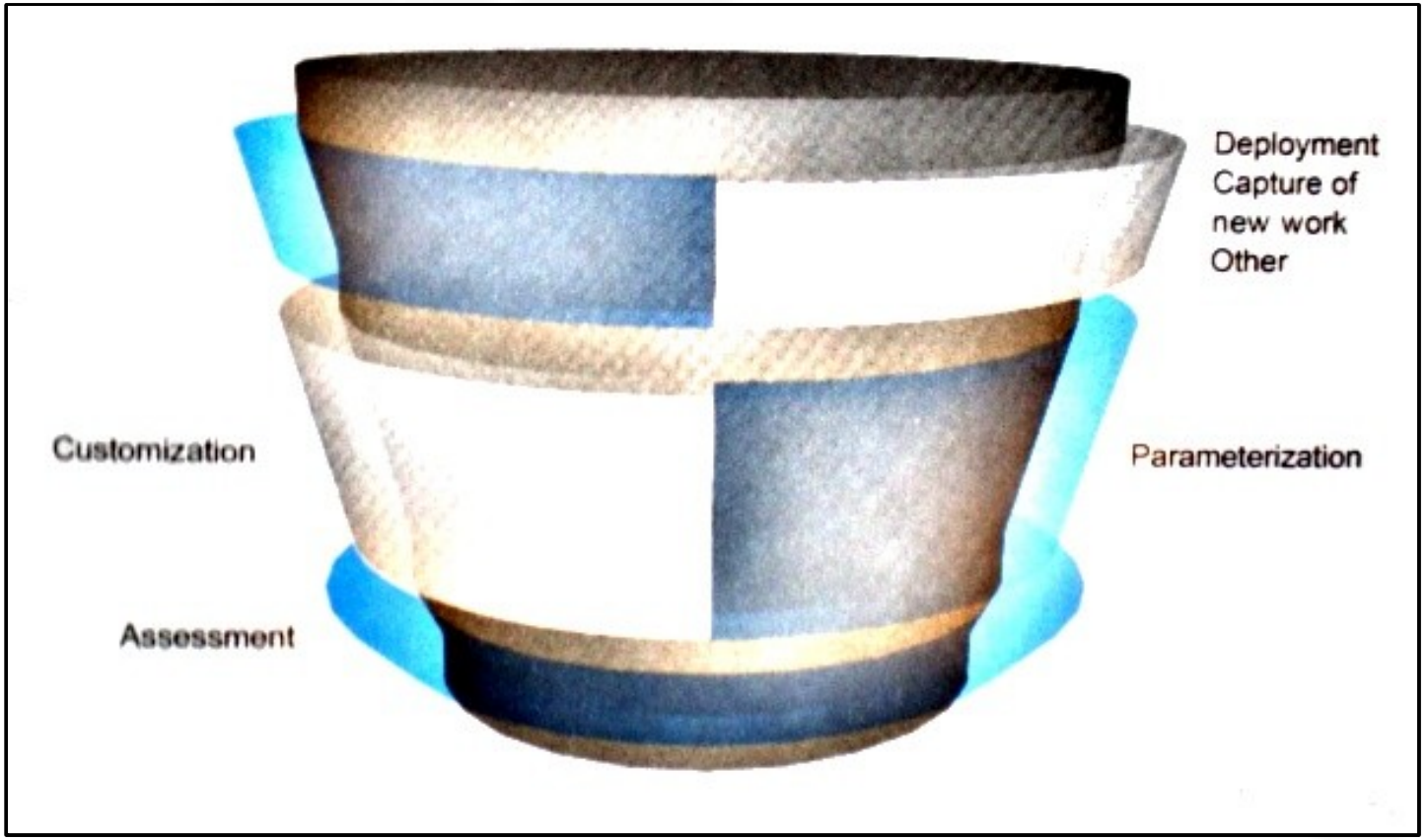


Processos de Desenvolvimento

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

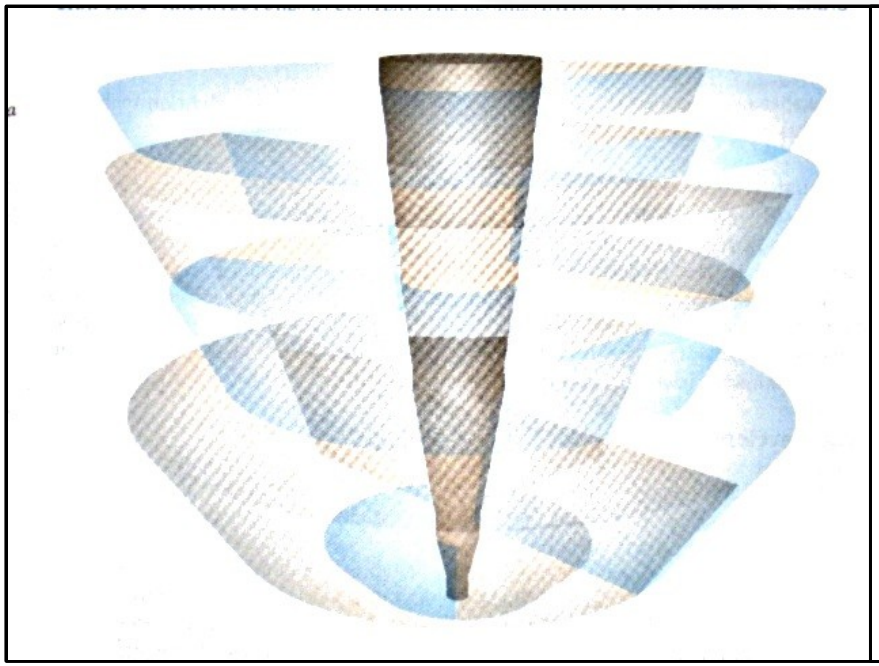
Processos de Desenvolvimento

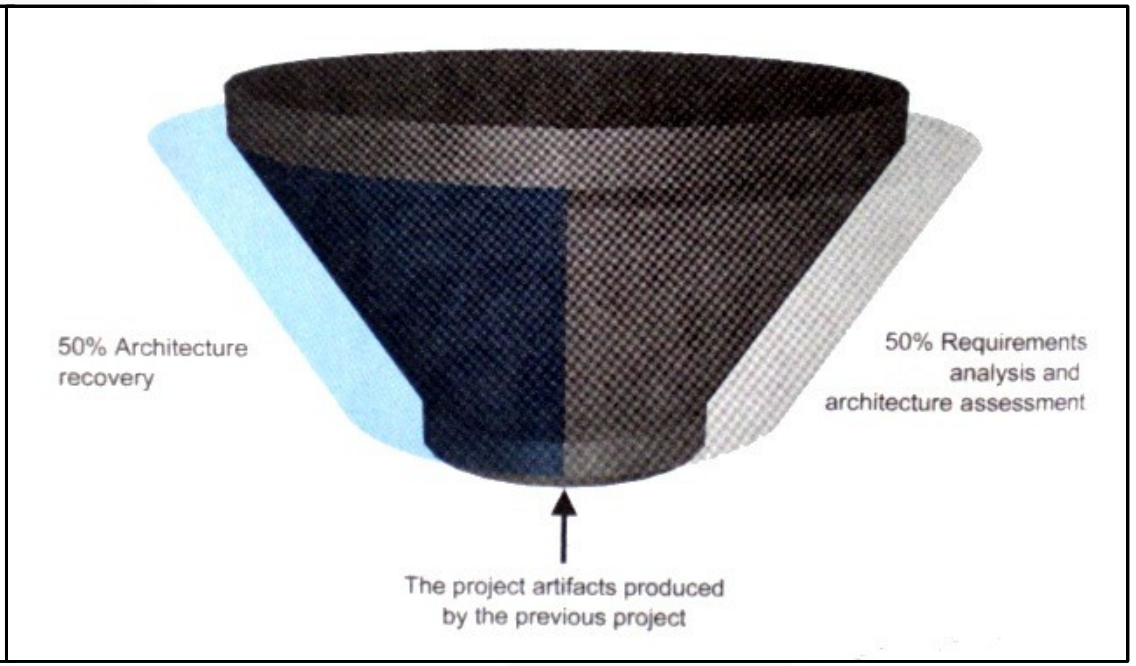
- Exemplo de descrição de processo:
  - *Domain-Specific Software Architecture* (DSSA)



Processos de Desenvolvimento

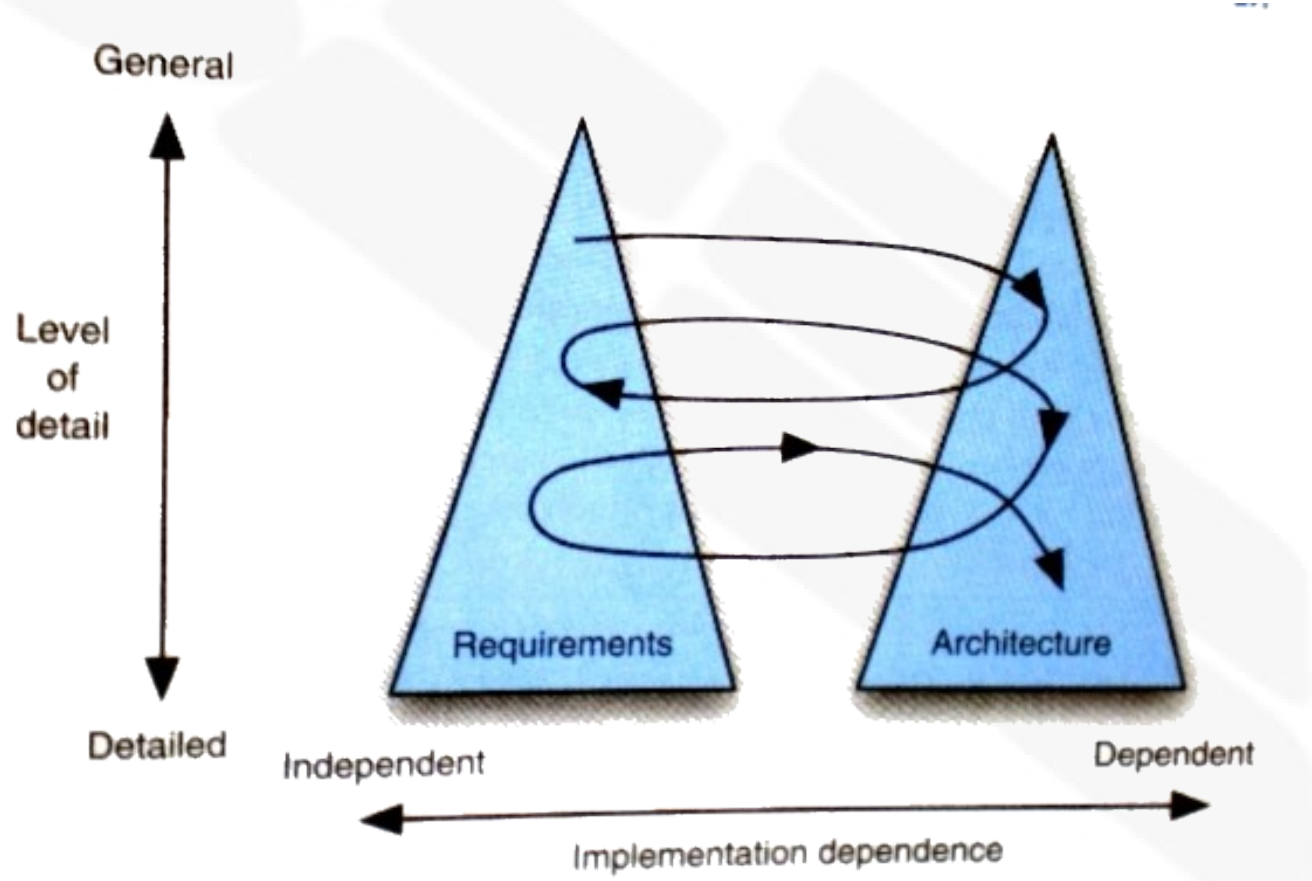
- Exemplo de descrição de processo:
  - Desenvolvimento ágil





Processos de Desenvolvimento

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



INF016 – Arquitetura de Software

02 – Reorientação da Engenharia de Software

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