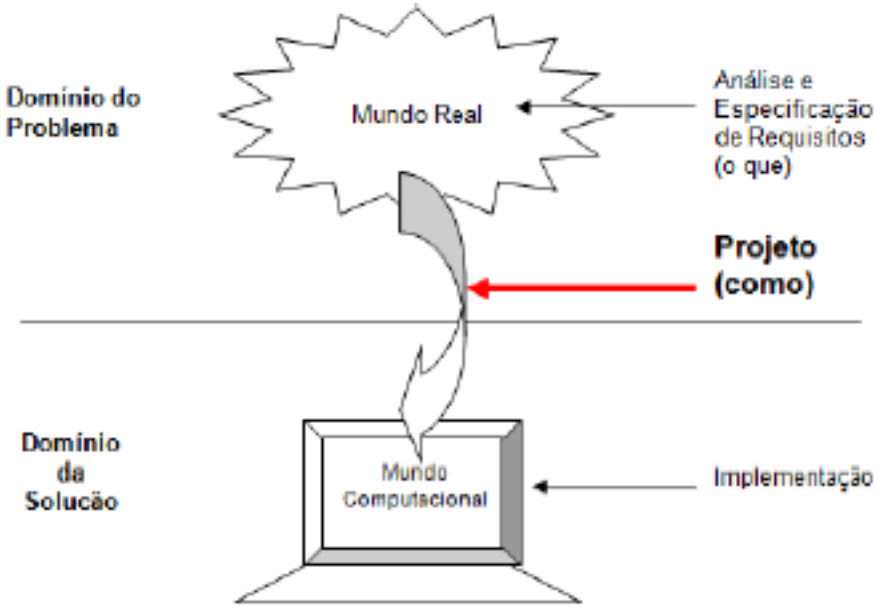
Projeto – Conceitos Iniciais

Definição

A etapa de **Projeto** tem início assim que os requisitos do software tiverem sido modelados e especificados (pelo menos parcialmente).

É a última atividade de modelagem.

É a primeira atividade que leva em conta aspectos tecnológicos.



– Princípios de Projeto

Em geral, um modelo de projeto (por exemplo, uma planta arquitetônica de uma casa) deve:

- começar representando a totalidade da coisa a ser construída;
- refinar para prover orientação para a construção de cada detalhe;
- prover diferentes visões da coisa.

Mais especificamente, o projeto de software deve:

- considerar abordagens alternativas com base nos requisitos do problema, recursos disponíveis e conceitos de projeto;
- ser rastreável ao modelo de análise;
- não "reinventar a roda", isto é, reutilizar componentes;
- minimizar a distância conceitual e semântica entre o software e o mundo real;
- exibir uniformidade (estilo) e integração (interfaces entre componentes);
- ser estruturado para acomodar mudanças (alterabilidade);
- acomodar circunstâncias não usuais e, se necessário abortar o processamento, fazê-lo de modo elegante;
- apresentar nível de abstração superior ao código fonte. Projeto não é codificação;
- Ser passível de avaliação da qualidade;
- Ser revisado para minimizar erros semânticos.

A Etapa de Projeto

Independente do paradigma adotado, a etapa de Projeto inclui definir:

Projeto da Arquitetura do Software: visa definir os elementos estruturais do software e seus relacionamentos.

Projeto dos Elementos da Arquitetura: visa projetar em um maior nível de detalhes cada um dos elementos estruturais definidos na arquitetura.

Projeto de Interfaces: tem por objetivo descrever como deverá se dar a comunicação entre os elementos da arquitetura (interfaces internas), a comunicação do sistema em desenvolvimento com outros sistemas (interfaces externas) e com as pessoas que vão utilizá-lo (interface com o usuário).

Projeto Detalhado: visa refinar e detalhar a descrição procedimental e das estruturas de dados dos elementos mais básicos da arquitetura do software.

Considerando o **paradigma orientado a objetos**, fazem parte da arquitetura de um sistema:

Lógica de Domínio: é o elemento da arquitetura que trata de toda a lógica do sistema, englobando tanto aspectos estruturais (classes de domínio derivadas dos modelos conceituais estruturais da fase de análise), quanto comportamentais (classes de processo que tratam das funcionalidades descritas pelos casos de uso).

Interface com o Usuário: é o elemento da arquitetura que trata da interação humano-computador. Envolve tanto as interfaces propriamente ditas (objetos gráficos responsáveis por receber dados e comandos do usuário e apresentar resultados) quanto o controle da interação, abrindo e fechando janelas, habilitando ou desabilitando botões etc.

Persistência: é o elemento da arquitetura responsável pelo armazenamento e recuperação de dados em memória secundária (classes que representam e isolam os depósitos de dados do restante do sistema).

Qualidade de Produto de Software

- **Funcionalidade**: refere-se à existência de um conjunto de funções que satisfaz às necessidades explícitas e implícitas e suas propriedades específicas. Tem como sub-características: adequação, acurácia, interoperabilidade, segurança de acesso e conformidade.
- **Confiabilidade**: diz respeito à capacidade do software manter seu nível de desempenho, sob condições estabelecidas, por um período de tempo. Tem como sub-características: maturidade, tolerância a falhas, recuperabilidade e conformidade.
- **Usabilidade**: refere-se ao esforço necessário para se utilizar um produto de software, bem como o julgamento individual de tal uso por um conjunto de usuários. Tem como sub-características: inteligibilidade, apreensibilidade, operacionalidade, atratividade e conformidade.
- **Eficiência**: diz respeito ao relacionamento entre o nível de desempenho do software e a quantidade de recursos utilizados sob condições estabelecidas. Tem como sub-características: comportamento em relação ao tempo, comportamento em relação aos recursos e conformidade.
- **Manutenibilidade**: concerne ao esforço necessário para se fazer modificações no software. Tem como sub-características: analisabilidade, modificabilidade, estabilidade, testabilidade e conformidade.
- **Portabilidade**: refere-se à capacidade do software ser transferido de um ambiente para outro. Tem como sub-características: adaptabilidade, capacidade para ser instalado, coexistência, capacidade para substituir e conformidade.

Qualidade de Produto

- Completeza
- Desempenho
- Segurança contra acessos indevidos
- Facilidade de uso
- Confiabilidade
- Mantenibilidade
- Economia

Tecnologia Imperfeita e Requisitos Tecnológicos

Em função das limitações da tecnologia, devemos tomar decisões que adicionam os requisitos tecnológicos à essência do sistema.

Limitações da tecnologia:

- Custo
- Capacidade de armazenamento
- Velocidade de processamento
- Aptidão dos processadores: processador matemático, para comunicação entre processadores, etc.
- A tecnologia é passível de falha.

Impactos da tecnologia imperfeita:

- Uso de processadores diferentes / Distribuição / Comunicação
- Redundância: repetição de atividades e dados; inclusão de dados derivados (por exemplo, totalizadores)
- Novas atividades e funções acrescidas em função da imperfeição da tecnologia.

Levantamento dos Requisitos Tecnológicos junto aos Usuários:

- Localização geográfica de usuários / Transporte de dados
- Problemas operacionais nas atividades dos usuários
- Definição do ambiente de hardware e software de produção:
  - Freqüência de disparo das atividades
  - Volume de dados (inicial, estimativa de crescimento, política de esvaziamento)
  - Tempo de resposta
  - Restrições técnicas (novo ambiente?)
  - Restrições ambientais (temperatura, etc.)
  - Requisitos de confiabilidade (tempo mínimo entre falhas)
  - Restrições de segurança (classes de usuários e acesso)
  - Interface com o usuário

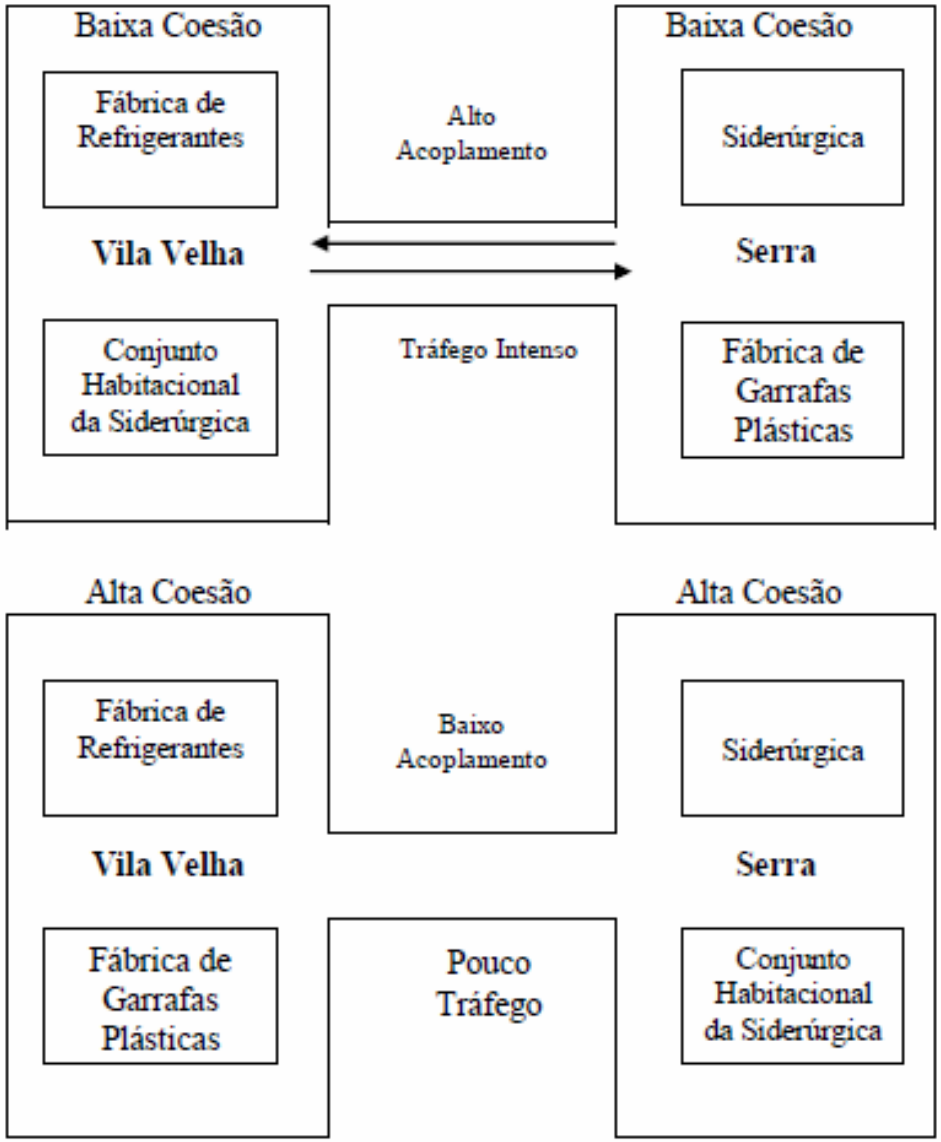
Qualidade do Projeto de Software

Um bom projeto de software deve apresentar determinadas características de qualidade, tais como facilidade de entendimento, facilidade de implementação, facilidade de realização de testes, facilidade de modificação e tradução correta das especificações de requisitos e de análise (PFLEEGER, 2004). Para se obter bons projetos, é necessário considerar alguns aspectos intimamente relacionados com a qualidade dos projetos, dentre eles (PRESSMAN, 2006):

- **Níveis de Abstração**: a abstração é um dos modos fundamentais pelos quais os seres humanos enfrentam a complexidade. Assim, um bom projeto deve considerar vários níveis de abstração, começando com em um nível mais alto, próximo da fase de análise. À medida que se avança no processo de projeto, o nível de abstração deve ser reduzido. Dito de outra maneira, o projeto deve ser um processo de refinamento, no qual o projeto vai sendo conduzido de níveis mais altos para níveis mais baixos de abstração.
- **Modularidade**: um bom projeto deve estruturar um sistema como módulos ou componentes coesos e fracamente acoplados. A modularidade é o atributo individual que permite a um projeto de sistema ser intelectualmente gerenciável. A estratégia "dividir para conquistar" é reconhecidamente útil no projeto de software, pois é mais fácil resolver um problema complexo quando o mesmo é dividido em partes menores gerenciáveis

- **Ocultação de Informações**: o conceito de modularidade leva o projetista a uma questão fundamental: até que nível a decomposição deve ser aplicada? Em outras palavras, quão modular deve ser o software? O princípio da ocultação de informações sugere que os módulos / componentes sejam caracterizados pelas decisões de projeto que cada um deles esconde dos demais. Módulos devem ser projetados e especificados de modo que as informações neles contidas (dados e algoritmos) sejam inacessíveis a outros módulos, sendo necessário conhecer apenas a sua interface. Ou seja, a ocultação de informação trabalha encapsulando detalhes que provavelmente serão alterados independentemente em diferentes módulos. A interface de um módulo revela apenas aqueles aspectos considerados improváveis de mudar (BASS; CLEMENTS; KAZMAN,
- **Independência Funcional**: a independência funcional é uma decorrência direta da modularidade e dos conceitos de abstração e ocultação de informações. Ela é obtida pelo desenvolvimento de módulos com finalidade única e pequena interação com outros módulos, isto é, módulos devem cumprir uma função bem estabelecida, minimizando interações com outros módulos. Módulos funcionalmente independentes são mais fáceis de entender, desenvolver, testar e alterar. Efeitos colaterais causados pela modificação de um módulo são limitados e, por conseguinte, a propagação de erros é reduzida. A independência funcional pode ser avaliada usando dois critérios de qualidade: coesão e acoplamento. A coesão se refere ao elo de ligação com o qual um módulo é construído. Uma classe, p.ex., é dita coesa quando tem um conjunto pequeno e focado de responsabilidades e aplica seus atributos e métodos especificamente para implementar essas responsabilidades. Já o acoplamento diz respeito ao grau de interdependência entre dois módulos. O objetivo é minimizar o acoplamento, isto é, tornar os módulos tão independentes quanto possível.

Idealmente, classes de projeto em um subsistema deveriam ter conhecimento limitado de classes de outros subsistemas. Coesão e acoplamento são interdependentes e, portanto, uma boa coesão deve conduzir a um pequeno acoplamento. A Figura 6.2 procura ilustrar esse fato.



Diretrizes de Qualidade de Projeto

- Pressmann:
- Um projeto: (1) deve exibir uma arquitetura que tenha sido criada usando estilos e padrões arquiteturais (2) é composto de componentes bem projetados (3) deve poder ser implementado sido implementado de forma evolucionária
- O projeto deve ser modular
- O projeto deve conter representações distintas de dados, arquitetura,interfaces e componentes

- Um projeto deve levar a estruturas de dados que sejam apropriadas às classe a serem implementadas
- Um projeto deve levar à independência funcional
- Um projeto deve levar a interfaces que reduzam o acoplamento
- A notação utilizada no projeto deve ser adequada