Projeto de Software

Silvia Regina Vergilio - UFPR

Introdução ao Projeto

1. Objetivos
2. Importância
3. Fundamentos
4. O processo de projeto
5. Métodos de projeto
6. Analisando a estrutura do software

1) Objetivos

Definição (*Sommerville*): fase que visa a projetar uma estrutura de software que realiza a especificação

Modelos de representação do domínio (modelos de análise)

⇓ *Aplica Técnicas e princípios*

Detalhe suficiente para permitir a realização física do sistema (implementação)

Análise e Projeto - Diferenças

Análise — "o quê"

Modelagem do problema (investigação para entendimento do problema)

Quais os processos de negócio relacionados com o seu uso?

Projeto — "como"

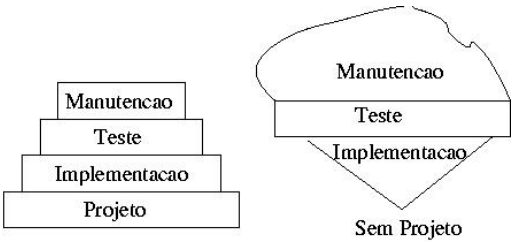
Modelagem da solução (criação)

Como exatamente o software irá capturar e registrar informações?

2) Importância

- Fase na qual a qualidade é criada e poderá ser efetivamente avaliada.
- Servirá como fundamento para as fases de codificação, teste e manutenção.

Importância



Importância

" ... tentamos resolver o problema passando rapidamente pelo processo de projeto para que sobre tempo suficiente no final para detectar e corrigir defeitos que foram introduzidos porque dedicamos pouco tempo ao projeto."

3) Fundamentos (Princípios)

- Particionamento (decomposição; divisão e conquista)
- Abstração
- Encapsulamento (ocultamento da informação)
- Modularidade
- Separação de preocupações (concerns)
- Separação de políticas da execução de algoritmos
- Acoplamento e coesão

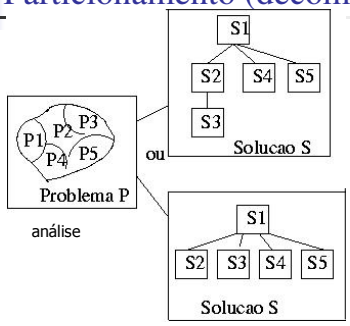
Particionamento (decomposição)

• Um problema P complexo deve ser decomposto em sub-problemas menores (ou sub-problemas) que devem ser entendidos e solucionados separadamente. A combinação das sub-soluções resolve o problema original.

• Pode haver muitas maneiras de combinar as soluções

→ um problema P pode ser solucionado por muitas estruturas candidatas – diferentes arquiteturas

Particionamento (decomposição)



Abstração

• Essencial para se lidar com complexidade.

• Podem existir vários níveis de abstração

• Recomenda que um elemento de projeto deve ser representado apenas por suas características essenciais, ou seja, detalhes desnecessários são descartados, de forma a permitir:

- sua distinção de outros elementos por parte do observador.
- uma representação o mais simples possível,
- facilidade de entendimento, comunicação e avaliação.

Encapsulamento

- Ocultamento da Informação: princípio que diz que módulos devem ser especificados e projetados de modo que a informação (dados ou procedimento) nele contida seja inacessível a outros módulos que não tenham necessidade daquela informação
- A interface definida de forma a revelar o mínimo possível sobre o seu funcionamento interno.
- Cada componente do programa deve conter uma única decisão de projeto.

Encapsulamento

- Reduz efeitos colaterais, facilita reuso, manutenção e testes, entendimento, etc.
- Acessam-se dados de um objeto apenas com métodos do próprio objeto que funcionam como interface para outros objetos.
- Pode ser obtido de diferentes maneiras: modularizando o sistema, separando suas preocupações, separando, interfaces de implementações, ou separando políticas da execução de algoritmos.

Modularidade

- Característica do software que permite a sua inteligibilidade, permite dividí-lo em componentes menores, chamados **módulos.**
- Vantagens:
  - Facilita o entendimento, uma vez que cada módulo pode ser estudado separadamente;
  - Facilita o desenvolvimento, uma vez que cada módulo pode ser projetado, implementado e testado separadamente;

Modularidade

- Vantagens (cont)
  - Diminui o tempo de desenvolvimento, uma vez que módulos podem ser implementados em paralelo, ou ainda reúsados; e
  - Promove a fexibilidade no produto, uma vez que um módulo pode ser substituído por outro, desde que implemente as mesmas interfaces.
  - Leva a um esforço menor para resolver problemas

C(x)=complexidade do problema
E(x)=esforço para resolver o problema
C(P1) > C(P2) ➔ E(P1) > E(P2)
C(P1+P2) >C(P1)+C(P2) ➔ E(P1+P2)>E(P1) + E(P2)

Modularidade

Como saber se eu dividi o suficiente?

R: Relação com outros princípios:

Modularidade efetiva se consegue com um bom particionamento, alta abstração, boa separação de preocupações, alta coesão e baixo acoplamento.

Separação de preocupações

- Regra: para definir os módulos de um sistema, preocupações diferentes ou não-relacionadas devem se restringir a módulos diferentes.
- Isto leva a alta coesão e baixo acoplamento

Separação de Decisões de Execução de Algoritmos

- Derivada da separação de preocupações.
- Um módulo deve se preocupar com as decisões sensíveis ao contexto do problema ou com a execução de algoritmos, mas não com ambas.
- Módulos específicos tomam decisão sobre dados e passam parâmetros para outros módulos que executam algoritmos sobre estes dados (por ex. Ordenação).
- Facilita o reuso e manutenção. Os módulos de execução de algoritmos ficam menos específicos.

Coesão e Acoplamento

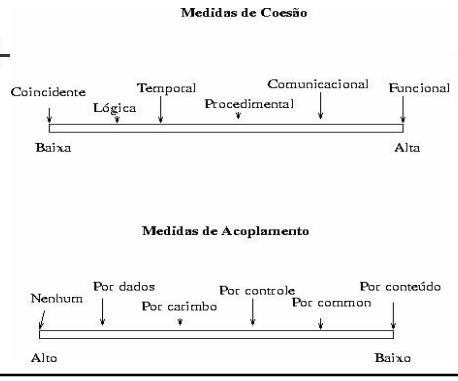
Baseadas nos princípios de um bom projeto: (Yourdon, Constantine e Myers)

- **Coesão**: medida da funcionalidade de um módulo; o quanto ele realiza uma tarefa específica
- **Acoplamento**: mede o grau de interdependência entre módulos

Coesão e Acoplamento

- Acoplamento (Pressman)
  - "O acoplamento é uma medida da interconexão entre os módulos de uma estrutura de software, depende da complexidade de interface entre módulos";
- Coesão
  - "Um módulo coeso executa uma única tarefa dentro do procedimento de software, exigindo pouca interação com procedimentos executados em outras partes de um programa."

Coesão e Acoplamento



Tipos de Coesão (deve ser alta)

- coincidente: as tarefas estão agrupadas sem qualquer critério. Não é possível relacionar funções.
- lógica: as funções estão relacionadas logicamente, geralmente existe um flag de controle para determinar a ordem a ser executada.
- temporal: o módulo realiza mais que uma função que devem ocorrer no mesmo intervalo de tempo

Tipos de Coesão

- procedimental: o módulo realiza mais que uma função, de tal forma que elas estão relacionadas a um procedimento geral e seguem uma ordem específica
- comunicacional: o módulo realiza funções que manipulam a mesma estrutura de dados
- funcional: o módulo realiza uma única função bem definida

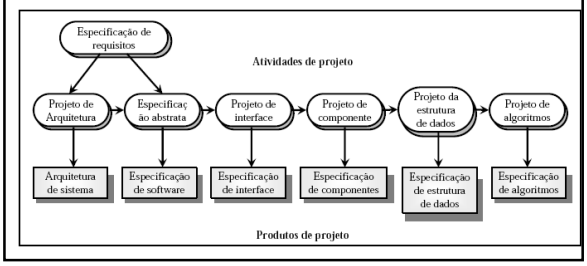
Tipos de Acoplamento (deve ser baixo)

- acoplamento por conteúdo: x faz referência direta ao interior de y
- acoplamento por common: x e y referenciam variáveis globais
- acoplamento por controle: x e y se comunicam por parâmetros sendo que um deles é um flag que controla o comportamento de um dos módulos

Tipos de Acoplamento

- acoplamento por carimbo ``stamp'': x e y se comunicam por parâmetros, sendo um deles, uma estrutura de dados
- acoplamento por dados: x e y se comunicam por parâmetros
- sem acoplamento: não existe dependência

4) O Processo do projeto



Principais Atividades

- Projeto da Arquitetura
  - do sistema
  - do controle
  - dos módulos
- Projeto dos Dados
- Projeto dos Procedimentos
- Projeto da Interface com outros Elementos do Sistema
- Projeto da Interface com o Usuário

Passos

- **Projeto Geral ou Preliminar**: fase que traduz a especificação do sistema em termos da arquitetura de dados e de módulos
- **Projeto Detalhado**: refinamento visando à codificação e especificação dos programas

Fase de refinamento da arquitetura (high-level design)

- Definição de pacotes (módulos), interfaces entre pacotes
- Decisão sobre uso/criação de bibliotecas e/ou componentes

Fase de projeto detalhado (low-level design)

- Atribuição de responsabilidades entre os objetos
- Construção de diagramas de classes
- Pode incluir documentação javadoc (ideal)
- Construção de diagramas de interação (opcional)
- Levantamento de necessidades de concorrência
- Considerações de tratamento de defeitos
- Detalhamento do formato de saída (interface com usuário, relatórios, transações enviadas para outros sistemas, ...)
- Definição do esquema do BD: mapeamento de objetos para tabelas se o BD for relacional

5) Métodos de projeto

- Um método governa a execução de alguma atividade seguindo abordagens rigorosas, disciplinadas e sistematizadas.
- Método de projeto: abordagem sistemática para desenvolver um projeto de software
- O projeto é normalmente documentado como um conjunto de modelos gráficos
-

Métodos de projeto

- Métodos Orientados por estrutura de dados
- Métodos Orientado por funções
- Métodos Orientado por objetos
- Métodos formais

Métodos orientados por estrutura de dados

- A representação de software é obtida das estruturas de dados.
- Mapear a representação da estrutura de dados para uma hierarquia de controle de software
- Refinar a hierarquia
- Desenvolver uma descrição procedimental das funções

Métodos: Jackson ("Jackson System Development")
Warnier-Orr (DSSD)

Métodos orientados por funções

- De Marco, 1979
- Gane e Sarson, 1982
- Análise estruturada moderna - inclui aspectos de sw de tempo real - Yourdon, 1990
- Análise essencial, McMenamin e Palmer, 1984
- Ward e Mello, 1985

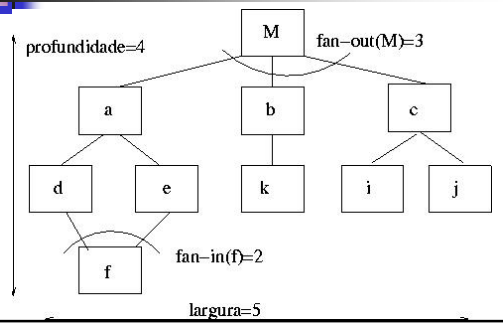
Métodos orientados a objeto

- Shlaer e Mellor, 1990
- Wirfs- Brook, 1991
- Coad-Yourdon, 1991
- Martin-Odel, 1994
- Rumbaugh, (OMT) 1994
- Book, 1994
- Jacobson, (OOSE) 1994
- Colleman, Fusion, 1996

Métodos formais

- Atividades para o desenvolvimento e verificação de sistemas utilizanod modelos formais,e linguagens de especificação formal (que oferecem rigor matemático) – (lg formais tais como: lg Z, VDL).
- Facilitam prova de correção e eliminam ambiguidades.
  - Ex: Vienna,
    CSP (Processos sequenciais comunicantes)
    Larch

6) Analisando a estrutura do software



Analisando a estrutura do software

- Módulo que controla um módulo: superior
  Módulo controlado por outro: subordinado
- Largura – abertura (amplitude) da estrutura
  Profundidade – número de níveis de controle

Analisando a estrutura do software

- Fan-out: número de módulos controlados por um dado módulo
  Fan-in: número de módulos que controlam um dado módulo
- Visibilidade – conjunto de componentes invocados ou utilizados até mesmo indiretamente
  Conectividade: conjunto de componentes diretamente invocados ou utilizados

Tipos de Módulos

Quanto ao tempo de incorporação

- macro – incluído em tempo de compilação
- subrotina ou procedimento – geração da seqüência de chamadas pelo compilador e posterior ligação.
- ligação dinâmica – em tempo de execução.

Tipos de Módulos

Quanto aos mecanismos de ativação

- referência – chamada de sub-programa
- interrupção – programa em execução é interrompido para execução de outro módulo

Tipos de Módulos

Quanto ao padrão de execução

- módulos convencionais – um ponto de entrada e um de saída, execução sequencial e completa
- módulos reentrantes – não possui dados locais, mais de um ponto de entrada, usado simultaneamente por mais de uma tarefa (compartilhados)

Tipos de Módulos

Quanto ao relacionamento com outros módulos

- módulo sequencial – referenciados e executados sem interrupção
- módulo incremental (corotinas) – pode ser interrompido antes do seu término e posteriormente reiniciado
- módulo paralelo (conrotinas) – executados simultaneamente com outros módulos em ambientes concorrentes