UNIVERSIDADE FEDERAL DO PARANÁ - UFPR
Bacharelado em Ciência da Computação

SOFT
DISCIPLINA: Engenharia de Software **DATA: _____/_____/_____**
AULA NÚMERO: 10 **PROFESSOR: Andrey**

APRESENTAÇÃO

O objetivo desta aula é apresentar e discutir os conceitos de coesão e acoplamento.

DESENVOLVIMENTO

Projetar *software* é uma atividade complexa. Grande parte desta complexidade é inerente à própria natureza do *software*. Sistemas de *software* são mais complexos proporcionalmente ao seu tamanho, que qualquer outra construção humana. Computadores são algumas das máquinas mais complexas já construídas pelo homem. Eles possuem um número muito grande de estados. Sistemas de *software* possuem algumas ordens de grandeza a mais de estados que computadores.

Além de serem complexos por natureza, sistemas de *software* são constantemente sujeitos a pressões por mudança. A maior pressão por mudanças vem da necessidade de modificação da funcionalidade do sistema que é incorporada pelo *software*. Outra razão para essa pressão por mudanças se deve ao fato do *software* ser uma entidade flexível e que pode ser mudado com muito mais facilidade do que produtos manufaturados.

Projeto de *Software*

O projeto de um sistema de *software* tem muitas características em comum com projetos de engenharia, do ponto de vista do mapeamento dos requisitos funcionais em soluções tecnológicas. “Projetar sistemas de *software* significa determinar como os requisitos funcionais são implementados na forma de estruturas de *software*” [PETERS; PEDRYCZ, 2001]. O conjunto dos requisitos para o produto de *software* são mapeados no conjunto das soluções tecnológicas para a construção do *software*.

Neste caso, as soluções tecnológicas são estruturas de *software*, necessárias para a codificação do sistema de *software*. Em outras palavras, “...projeto é a única maneira com a qual se consegue traduzir com precisão os requisitos do cliente em um produto de *software* ou sistema finalizado”. [PRESSMAN, 2005].

O processo de desenvolvimento de *software* está inserido no ciclo de vida do *software*. O padrão do *Institute of Electrical and Electronics Engineers* (IEEE), IEEE 1074.1-1997 para a criação de modelos de ciclo de vida de *software* estabelece que os principais processos da fase de desenvolvimento de *software* são: Requisitos, Projeto e Implementação [IEEE, 1997]. Nos modelos de ciclo de vida, a atividade de projeto recebe o resultado da atividade de requisitos e gera um modelo para a atividade de implementação ou codificação.

Durante o projeto de *software* é criada a estrutura interna de um sistema de *software*. Esta estrutura interna é chamada de arquitetura. Arquitetura de *software* pode ser definida como a estrutura ou organização dos componentes do programa (módulos), a maneira com a qual estes componentes interagem e a estrutura da informação que é usada por estes componentes [PRESSMAN, 2005].

O grande objetivo de um projeto de *software* é levar a um produto de *software* que tenha qualidade. Os principais quesitos de qualidade de *software* estão relacionados na Seção 1.1. Para conseguir alcançar os objetivos de qualidade do produto de *software* gerado, um projeto de *software* deve ter qualidade. Existem diretrizes para a qualidade de um projeto de *software*, e entre elas estão [PRESSMAN, 2005]:

1. Um projeto deve ser modular;

2. Um projeto deve conter representações distintas para dados, arquitetura, interfaces e componentes;
3. Um projeto deve levar a componentes que possuam características de independência funcional;
4. Um projeto deve levar a interfaces que reduzam a complexidade das conexões entre os componentes e o ambiente externo.

Diretrizes de qualidade como as citadas acima podem ser usadas como parâmetro para avaliar a qualidade de um projeto de *software*. Mas para atingir essas diretrizes, é fundamental que o projetista tenha disciplina na aplicação de um processo de desenvolvimento, além de seguir princípios de projeto de *software*. Existem princípios de projeto de *software* que um projetista deve seguir para gerar um sistema de *software* com qualidade. Pressman (2005) enuncia princípios de projeto de *software*, que são [PRESSMAN, 2005]:

1. O projeto deve ser rastreável para o modelo de análise;
2. Sempre considere a arquitetura do sistema a ser construído;
3. O projeto dos dados é tão importante quanto o projeto das funções de processamento;
4. As interfaces (ambas internas e externas) devem ser projetadas com cuidado;
5. A interface com o usuário deve estar sintonizada com as necessidades do usuário final;
6. O projeto, no nível dos componentes, deve ser funcionalmente independente;
7. Componentes devem ser fracamente acoplados entre eles e com o ambiente externo;
8. As representações (modelos) de projeto devem ser entendidas facilmente;
9. O projeto deve ser desenvolvido iterativamente.

Independência Funcional

Para tentar administrar essa complexidade inerente de sistemas de *software*, emergiu um conceito chamado Independência Funcional. Este conceito está intimamente ligado à modularidade, ocultação de informações e abstração. Em sistemas de *software*, a Independência Funcional pode ser medida através de dois critérios: coesão e acoplamento.

- "Coesão é uma medida da força funcional relativa de um módulo". Em outras palavras a coesão mede o grau com que as tarefas executadas por um único módulo se relacionam entre si. Para *software* orientado a objetos, coesão também pode ser conceituada como sendo o quanto uma classe encapsula atributos e operações que estão fortemente relacionados uns com os outros.
- "Acoplamento é uma medida da interdependência relativa entre os módulos". Para sistemas de *software* orientados a objetos pode-se definir acoplamento como sendo o grau com o qual classes estão conectadas entre si. Existem métricas definidas na literatura para coesão de um módulo e acoplamento entre módulos.

As métricas de coesão e acoplamento, tratam de avaliar como os componentes dependem uns dos outros. Neste caso, não são considerados como os requisitos funcionais, que levaram à criação destes componentes, são dependentes uns dos outros e nem se o conjunto de requisitos funcionais escolhido é apropriado. Quando se escolhe um conjunto de requisitos funcionais muito interdependente, os componentes da solução gerada tendem a ser interdependentes também.

O desenvolvimento de *software* orientado a objetos tem como um dos seus preceitos aumentar a coesão e diminuir o acoplamento entre os módulos do sistema. O desenvolvimento orientado a objetos facilita para o desenvolvedor criar componentes mais reutilizáveis. Para isso, a orientação a objetos disponibiliza abstrações como classes, objetos, interfaces, atributos e métodos. Além disso, a orientação a objetos introduziu conceitos como encapsulamento, herança e polimorfismo para aumentar a reutilização e a extensibilidade e facilitar a manutenção de sistemas de *software*.

Coesão

Uma classe com baixa coesão faz muitas coisas não relacionadas e leva aos seguintes problemas:

- Difícil de entender
- Difícil de reusar
- Difícil de manter
- "Delicada": constantemente sendo afetada por outras mudanças

Tipos de coesão

- Coincidente (pior)
- Lógico
- Temporal
- Procedural
- De comunicação
- Sequencial
- Funcional (melhor)

Coesão coincidental

- Há nenhuma (ou pouca) relação construtiva entre os elementos de um módulo
- No linguajar OO:
    - Um objeto não representa nenhum conceito OO
    - Uma coleção de código comumente usado e herdado através de herança (provavelmente múltipla)

Coesão lógica

- Um módulo faz um conjunto de funções relacionadas, uma das quais é escolhida através de um parâmetro ao chamar o módulo
- Semelhante a acoplamento de controle

Coesão temporal

- Elementos estão agrupados no mesmo módulo porque são processados no mesmo intervalo de tempo
- Exemplos comuns:
    - Método de inicialização que provê valores defaults para um monte de coisas diferentes
    - Método de finalização que limpa as coisas antes de terminar

Coesão procedural

- Associa elementos de acordo com seus relacionamentos procedurais ou algorítmicos
- Um módulo procedural depende muito da aplicação sendo tratada
- Junto com a aplicação, o módulo parece razoável
- Sem este contexto, o módulo parece estranho e muito difícil de entender

Coesão de comunicação

- Todas as operações de um módulo operam no mesmo conjunto de dados e/ou produzem o mesmo tipo de dado de saída
- Cura: isole cada elemento num módulo separado
- "Não deveria" ocorrer em sistemas OO usando polimorfismo (classes diferentes para fazer tratamentos diferentes nos dados)

Coesão sequencial

- A saída de um elemento de um módulo serve de entrada para o próximo elemento
- Cura: decompor em módulos menores

Coesão funcional (a melhor)

- Um módulo tem coesão funcional se as operações do módulo puderem ser descritas numa única frase de forma coerente
- Num sistema OO:
    - Cada operação na interface pública do objeto deve ser funcionalmente coesa
    - Cada objeto deve representar um único conceito coeso

Acoplamento

Com uma classe possuindo forte acoplamento, temos os seguintes problemas:

- Mudanças em uma classe relacionada força mudanças locais à classe
- A classe é mais difícil de entender isoladamente
- A classe é mais difícil de ser reusada, já que depende da presença de outras classes

Tipos de acoplamento

- Acoplamento de dados
- Acoplamento de controle

Acoplamento de dados

- Situações
    - Saída de um objeto é entrada de outro
    - Uso de parâmetros para passar itens entre métodos
- Ocorrência comum:

    - Objeto a passa objeto x para objeto b
    - Objeto x e b estão acoplados
- Uma mudança na interface de x pode implicar em mudanças a b

Acoplamento de controle

- Passar flags de controle entre objetos de forma que um objeto controle as etapas de processamento de outro objeto
- Ocorrência comum:
    - Objeto a manda uma mensagem para objeto b
    - b usa um parâmetro da mensagem para decidir o que fazer

Acoplamento de dados globais

- Dois ou mais objetos compartilham estruturas de dados globais
- É um acoplamento muito ruim pois está escondido
- Uma chamada de método pode mudar um valor global e o código não deixa isso aparente

Acoplamento de dados internos

- Um objeto altera os dados locais de um outro objeto
- Ocorrência comum:
    - Dados públicos, package visibility ou mesmo protected em java

ATIVIDADE

1. O que é Coesão?
2. O que é acoplamento?
3. Quais os tipos de coesão?
4. Quais os tipos de acoplamento?
5. Quais as vantagens em aumentar a coesão e diminuir o acoplamento entre os módulos de um sistema?

BIBLIOGRAFIA BÁSICA

PRESSMAN, R. S.. Engenharia de Software. Makron Books. 1995

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.. UML guia do usuário. Editora Campus. 2000.

BEZERRA, E.. Princípios de Análise e Projeto de Sistemas com UML. Editora Campus. 2003.

CARVALHO, A. M. B. R.; CHIOSSI, T. C. S.. Introdução à engenharia de software. Editora da Unicamp. 2001.