FUNDAMENTOS DO PROJETO DE SOFTWARE

O objetivo deste documento é passar ao leitor algumas noções básicas dos fundamentos do projeto de software, que são úteis para uma melhor compreensão do Projeto Orientado a Objetos. Alguns conceitos como abstração, modularização, omissão de informações, entre outros são fundamentais e são brevemente discutidos aqui.

O projeto é o primeiro passo da fase de desenvolvimento de qualquer produto ou sistema. O objetivo do projetista é construir um modelo (ou representação) de uma entidade que será construida. O processo utilisado no desenvolvimento do modelo combina intuição, julgamento baseado na experiência adquirida no desenvolvimento de entidades similares entre outros. Note que esses fatores não são precisos, isto é, não existe nenhuma fórmula que nos leve à uma entidade perfeita, e sim uma análise cuidadosa e complexa que nos dá pistas e informações sobre como proceder.

PROJETO E ENGENHARIA DE SOFTWARE

Projeto de software faz parte do núcleo técnico da engenharia de software, e é aplicado independente do paradigma de desenvolvimento empregado. É o primeiro das três atividades (Projeto, Código e Teste) necessárias para construir e checar um software. Cada uma dessas três atividades transforma informações de maneira que, por fim, resulta num produto (software) pronto e verificado. Nesta atividade (projeto) são necessários modelos para informação, funcionalidade e comportamento. Podemos usar vários métodos (no nosso caso, orientado a objeto) que descrevem maneiras diferentes de se administrar o projeto, mas todos resultam em três subprojetos que serão empregados na atividade de código. A saber:

- Projeto de Dados: Transforma a informação obtida durante a fase se análise (uma das fases iniciais do processo de software engineering) em estruturas de dados necessárias para a implementação do software;
- Projeto de Arquitetura: Define as relações entre os principais componentes estruturais do software;
- Projeto de Procedimento: Transforma os componentes estruturais em uma descrição procedural do software.

A partir desses subprojetos é gerado um código fonte do software que pode ser implementado e finalmente testado.
O Projeto, Código e Teste juntos absorvem cerca de 75% do custo (excluindo manutenção do software) de engenharia de software. É durante a primeira atividade (projeto) que são tomadas as decisões responsáveis pelo sucesso da implementação e pela facilidade com que o software será mantido (manutenção do software). É por isso que a atividade de projeto é considerada o pivô do Processo de Desenvolvimento.
A qualidade do software está diretamente ligada ao projeto , que é a única maneira de traduzir precisamente as exigências do cliente (consumidor) em um produto (software) final. O projeto d software serve de base para todos os processos de engenharia e manutenção de software que o seguem. Sem ele, arriscamos construir um produto instável, que pode falhar quando pequenas mudanças forem feitas, que será difícil de se testar e cuja qualidade só poderá ser avaliada no final do processo de engenharia, onde gera lmente o tempo é curto e os recursos (financeiros) escassos.

O PROCESSO DO PROJETO DE SOFTWARE

Projeto de software é um processo pelo qual transformamos uma dada informação em representação de software. Inicialmente a representação se assemelha a um "rascunho" de programa, mas após vários refinamentos obtemos uma representação que é muito próxima do código fonte do programa. Do ponto de vista de gerenciamento de projeto, o software design consiste de duas partes:

- Projeto Preliminar, que se responsbiliza pela transformação das informações em dados e estruturas de software.
- Projeto Detalhado, que foca os refinamentos da estrutura de programa e dados, detalhando passo a passo as estruturas e algoritmos.

Essas partes englovam inumeras atividades de projeto, como projeto de dados, arquitetura de procedimento, de interface, entre outras.

Qualidade do Projeto de Software

Durante o processo de projeto avaliamos a qualidade do mesmo. Para isso, precisamos estabelecer critérios para um bom projeto. Apresentamos aqui uma

simplificação de tais critérios:

1. Um projeto deve apreentar uma organização hierárquica que faça um uso inteligente da ordem entre os vários componentes do software;
2. Deve ser modular, ou seja, o software deve ser dividido em blocos e subblocos que realizam funções e subfunções específicas;
3. Deve conter representações distintas (e separáveis) de dados e procedimentos;
4. Deve levar à módulos que (subrotinas e procedimentos) que apresentem características funcionais independentes;
5. Deve levar à interfaces que reduzam a complexidade das conexões entre módulos e com o ambiente externo;
6. Deve ser derivado (refinado) por um método direcionado por informações obtidas durante o processo de análise de requisitos do software.

Essas características de um bom projeto não são obtidas por acaso, elas surgem das aplicações de princípios fundamentais de projeto, metodologia sistemática e revisão do projeto.

FUNDAMENTOS DO PROJETO

Um conjunto de conceitos fundamentais para o projeto de software têm evoluido nos últimos 30 anos. Apesar do grau de interesse em cada conceito variar durante os anos, eles resistiram ao tempo e agora representam conceitos sofisticados em que o projetista de software pode se basear durante o desenvolvimento de um software. Citaremos alguns desses conceitos:

Abstração

Quando consideramos uma solução modular para qualquer problema, vários níveis de abstração podem ser definidos partindo do inicio do trabalho (representação formal da solução) até o final (código fonte que representa a solução do problema).
Nos níveis mais altos de abstração, as representações propostas para a solução são expressas utilizando-se a linguagem do ambiente do problema. Nos níveis mais baixos, utilizamos uma linguagem mais procedural e computacional. Seguindo essa ordem, no nível mais baixo de abstração representamos a solução do problema através do código fonte.
Cada passo em um processo de software engineering é um refinamento (aprimoramento) do nível de abstração da solução do problema.

Simpificando, começamos propondo uma solução cuja representação é expressa através de termos familiares ao ambiente do problema, e progredimos propondo soluções cada vez mais procedurais, detalhando cada vez mais a representação da solução, diminuindo o nível de abstração, até que atingimos o nível mais baixo, que ocorre quando o código fonte do programa que representa a solução do problema, é gerado.
Conforme nos aprofundamos nos níveis de abstração, procuramos criar abstrações procedurais e de dados. Uma ABSTRAĆÃO PROCEDURAL é uma seq&uauml;ência (com nome) de instruções que possui uma função específica e limitada. Um exemplo simples poderia ser, num alto nível de abstração, a palavra "entrar" da expressão 'entrar por uma porta' pois "entrar" implica numa seq&uauml;ência de vários passos procedurais (caminhar até a porta, alcançar a maçaneta, virar a maçaneta, etc...).
Uma ABSTRAĆÃO DE DADOS é uma coleção (com nome) de dados que descreve um objeto de dados. Citamos como exemplo o "cadastro" de um cliente. Este objeto de dados (cadastro) é um conjunto de pequenas informações independentes ou dados referentes ao objeto (nome, endereço, preferências...).

Refinamento

Refinamento é um aprimoramento. quando dizemos que refinamos alguma coisa, queremos dizer a grosso modo que detalhamos essa coisa. Uma das formas de refinamento é o refinamento passo a passo (stepwise refinement), mostrado a seguir:

Refinamento Passo à Passo

É um modelo top-down recente de projeto, proposto por Niklaus Wirth, onde se desenvolve a arquitetura do software através de sucessivos níveis de refinamento dos procedimentos que o compõem. Uma hierarquia é formada pela decomposição (refinamento) de abstrações procedurais até que se chegue num modelo totalmente descrito por uma linguagem de programação. É normal e totalmente esperado que o refinamento dos procedimentos seja acompanhado em paralelo por um refinamento de dados.
Resumindo, refinamento é um processo de elaboração onde começamos com uma declaração de função (ou descrição de informação) que está definido em um alto nível de abstração, ou seja, a declaração descreve a função conceitualmente, sem muitas pistas sobre como ela trabalha ou como se estruturam os dados. O refinamento consiste de o projetista trabalhar essa declaração de função original de modo que em cada passo seja obtido mais e mais detalhes, diminuindo o nível de abstração, obtendo cada vez mais detalhes e informações sobre o funcionamento da função e da organização dos dados e de

suas estruturas.

Modularidade

Não é difícil perceber que um programa de grandes dimensões não modular (monolítico) não pode ser facilmente compreendido, e dependendo de suas dimensões, sua compreensão é quase impossível. Experiências na área de Software Engineering mostram que é mais fácil resolver problemas quando os dividimos em problemas menores. Seguindo essa linha de raciocínio, podemos extrapolar e dizer que se subdividirmos um software em infinitos módulos, a dificuldade em desenvolvê-lo será mínima!. Entretanto, como em quase tudo na engenharia, temos que considerar outros fatores concorrentes. Como já vimos, a dificuldade em se desenvolver um software diminui conforme aumentamos o número de módulos, porém a dificuldade em se criar uma interface entre módulos (forma como os módulos se comunicam e interagem) aumenta junto com o número de módulos criados, ou seja, tanto devemos evitar a não modularidade quanto a supermodularidade. A resposta está nesse meio termo.

Arquitetura de Software

Arquitetura de software engloba duas importantes características de um programa de computador:

- A Estrutura Hierárquica dos Procedimentos;
- A Estrutura de Dados.

A evolução do programa e de sua estrutura de dados começa com uma definição do problema, e termina quando cada parte do problema é resolvido por uma ou mais partes (módulos) do programa.
O problema pode ser resolvido por vários modos e consequentemente por várias estrturas diferentes, e não existe uma resposta simples para qual estrutura é melhor.

Controle Hierárquico (Estrutura do Programa)

Controle Hierárquico representa a organização dos componentes (módulos) do

programa.
Existem muitas notações para representar o controle hierárquico como os diagramas de Warnier-Orr e Jackson, mas os mais comuns são os diagramas baseados em estruturas de árvore (em Projeto Orientado a Objeto, o conceito de Controle Hierárquico não é tão óbvio).
Em Estrutura de Programa, um dado módulo A que controla outro módulo B é denominado "superordinate" do módulo B e consequentemente, o módulo B é "subordinate" do módulo A.
XXXXXXXXXX esplicar visibility e connectivity (pg 327) XXXXXXXXXXX

Estrutura de Dados

Estrutura de dados é uma representação das relações logicas existentes entre cada elemento individual de um dado. A estrutura de dados é tão importante quanto a estrutura do programa para a arquitetura do software porque inevitavelmente afeta o projeto procedural final.
A estrutura de dados dita a organização, métodos de acesso, grau de associatividade e alternativas de processamento para a informação.
Algumas das estruturas de dados denominadas clássicas são: Estrutura ligada (lista ligada), vetor seq&uauml;encial, array n dimensional, estrutura em árvore, etc.
É interessante notar que tanto estruturas de dados quanto de programas podem ser representados em diferentes níveis de abstração, por exemplo PILHA, que é um modelo conceitual de estrutura de dados que pode ser implementado por uma lista ligada ou por um vetor. Dependendo do nível de detalhe do projeto em que se está, o funcionamento "interno" da pilha pode ou não ser especificado.

Omissão de Informação (Information Hiding)

O conceito de modularidade nos leva a uma pergunta fundamental: Como devemos decompor uma solução (em software) para obtermos o melhor conjunto de módulos?. O princípio da omissão de informação nos ajuda nesta tarefa propondo que toda informação (dados e procedimentos) de um módulo seja acessível apenas para os módulos que realmente utilizem tais informações. O uso da omissão de informação como critério de projeto para sistemas modulares propcia os maiores benefícios quando modificações são necessárias dur ante a fase de teste ou na manutenção pois uma vez que erros introduzidos durante uma modificação tem menos chance de se propagar pelo software se a maioria dos dados e procedimentos estiver escondido de outras partes do software.
