DESENVOLVIMENTO DE SOFTWARE EM CAMADAS

1. Resumo

Este material procura descrever os principais conceitos envolvidos com a técnica de desenvolvimento de software em camadas, metodologia largamente aceita e utilizada no desenvolvimento de softwares corporativos.

2. Conceitos

Existem diferentes tipos de software para diferentes tipos de aplicações, cada uma com suas características e complexidades.

> *Aplicações que manipulam uma grande quantidade de dados que devem ser persistidos, às vezes por muitos anos, que são acessadas por uma grande quantidade de pessoas concorrentemente, que possuem uma quantidade significante de código e que se destina a mostrar esses dados de diferentes maneiras e com diferentes propósitos, geralmente precisam se integrar com outros sistemas, construídos em diferentes tempos com diferentes tecnologias e trabalham com as regras de negócio da instituição para a qual estão sendo desenvolvidos, que geralmente estão em constante mudança.*
>
> *(Martin Fowler, 2002)*

Por causa das características citadas acima, é visível que aplicações corporativas têm um alto grau de complexidade e precisam estar preparadas para sofrerem mudanças constantemente, por isso, muitos padrões foram criados para facilitar as mudanças nas estruturas e regras sobre as quais o software foi construído.

3. Camadas - principal padrão utilizado para aplicações corporativas

A maioria dos padrões utilizados no desenvolvimento de aplicações corporativas está relacionada com a separação do código em camadas, isto é, eles procuram definir como essas camadas estão organizadas e como é feita a comunicação entre elas.

3.1. Coesão e Acoplamento

Atualmente existem duas métricas básicas de qualidade que devem ser seguidas por um módulo de software (método, classe, *namespace* ou qualquer divisão que se faz em um software dependendo do tamanho do mesmo): coesão e acoplamento.

Módulos de software devem ter uma alta coesão, isto é, todos os componentes desse módulo **deveriam** ter as mesmas responsabilidades e, idealmente, cada módulo deveria está focado em resolver um problema específico. Além disso, devem ter um baixo acoplamento, ou seja, todo módulo deveria ser o mais independente possível de outro e uma alteração da implementação de um módulo não deveria afetar nenhum outro.

Um alto grau de acoplamento entre muitos objetos é sinal de um design pobre porque deixa o código difícil de ler e entender e, mais importante, torna o código pouco flexível a mudanças.

> *"O uso de Camadas é um padrão arquitetural que ajuda na tarefa de separar responsabilidades, promovendo baixo acoplamento e alta coesão em um sistema".*
>
> *(Philipe Calçado, 2005)*

Idealmente, Camadas se comunicam apenas com Camadas adjacentes.

3.2. Arquitetura em duas camadas

A necessidade de se separar uma aplicação em camadas surgiu no começo da década de 90 com os sistemas com arquitetura cliente-servidor. Existem geralmente duas camadas nesses sistemas: a interface gráfica com o usuário e a camada de persistência (geralmente um Banco de Dados relacional).

Há duas possibilidades de implementação para esses sistemas: "cliente rico" ou "cliente pobre" (conhecidas também como: "cliente gordo" ou "cliente magro").

3.2.1. Cliente rico

No cliente rico a lógica da aplicação está embutida junto com o código da interface gráfica com o usuário. Na medida em que os sistemas vão crescendo e se tornando mais complexos, essa técnica gera um grande número de problemas, como uma grande quantidade de código repetido. Uma pequena mudança pode ter um impacto enorme na aplicação (e isso é o que exatamente se quer evitar em sistemas corporativos).

3.2.2. Cliente pobre

A outra abordagem seria colocar a lógica da aplicação em *Stored Procedures* no banco de dados relacional. Mas essa solução também tem suas desvantagens como, por exemplo, a perda de portabilidade, a dificuldade de trabalhar com transações quando está envolvido mais de um SGDB e não se poder trabalhar com orientação a objetos e suas vantagens.

3.3. Arquitetura em três camadas

Diante dos problemas encontrados na Arquitetura 2 Camadas, surgiu a necessidade de se evoluir para a chamada programação em três camadas, na qual se permanece com a camada de interface gráfica com o usuário (também conhecida como camada de apresentação) e se manteve também

a camada de persistência, porém foi **criada uma nova camada** entre as duas anteriores, que modela o domínio da aplicação e onde ficam as regras de negócio da mesma; chamada por muitos autores de camada de negócio.

Com a introdução dessa camada, resolveu-se grande parte dos problemas citados anteriormente. As camadas de persistência e a de negócio nunca deveriam ser dependentes da camada de apresentação, pois essa dependência geraria muita duplicação de código, visto que os dados de uma aplicação podem ser mostrados em diferentes partes e de diferentes maneiras e, consequentemente, acarretaria muito trabalho para se alterar qualquer parte desse sistema.

A separação dessas três camadas não precisa ser fixa. Para sistemas simples podem ser três métodos diferentes, para sistemas mais complexos pode-se separar as três camadas em três classes diferentes. Com o aumento da complexidade, faz-se necessária a divisão das três camadas em três *assemblies* diferentes. A forma de realizar a separação não é única e depende de cada caso, mas o importante é que se separe o sistema em três camadas.

4. Vantagens e desvantagens

Abaixo estão listadas algumas vantagens e desvantagens em desenvolver sistemas utilizando uma arquitetura em camadas:

4.1. Vantagens

- Reduzem complexidade: agrupam componentes e simplificam a comunicação entre eles;
- Reduzem dependência/acoplamento: a regra de comunicação evita dependências diretas entre componentes de Camadas diferentes;
- Favorecem a coesão: componentes de responsabilidades relacionadas são agrupados;
- Promovem a reusabilidade: camadas podem ser reutilizadas em outros sistemas ou podem ser substituídas;
- É um padrão arquitetural conhecido: facilita a comunicação e entendimento entre desenvolvedores.

4.2. Desvantagens

- Apenas complicam um sistema muito simples: não é qualquer sistema que exige o uso de Camadas;
- Possibilidade de overdose: muitos arquitetos acabam criando Camadas demais e tornando a aplicação extremamente complexa.

5. Exemplo do uso de camadas

Nesta seção é apresentado um exemplo, para efeito de esclarecimento, da implicação do uso do padrão de desenvolvimento em camadas. É apresentada uma aplicação bem simples que foi projetada em uma arquitetura multicamadas.

5.1. Estrutura do projeto

Na Figura abaixo, é mostrada a estrutura do projeto desenhada para ter quatro camadas: Apresentação, Aplicação, Modelo e Persistência.

Explicando a figura ao lado, podemos ver no diretório "1. Apresentação" os projetos **Uniplac.eAgenda.WindowsApp, Uniplac.eAgenda.ConsoleApp e Uniplac.eAgenda.WebApp** que representam a camada de Apresentação do software, esta camada é responsável por construir a interface gráfica com o usuário e tratar seus eventos.

No segundo diretório mostra-se a camada de Aplicação, esta é a responsável por coordenar a sequência das tarefas do aplicativo.

A camada de Modelo, que possui uma modelagem do domínio do aplicativo está representada nesse exemplo pelo objeto Tarefa, nesse caso este objeto não possui Regras de Negócio, apenas os dados do aplicativo.

A camada de infraestrutura/persistência foi implementada usando-se o padrão DAO, que é um *design pattern* para acessar dados em um Banco de Dados (veremos mais adiante).